Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ МЕТОДУ КЛАСТЕРИЗІЦІЇ НА ОСНОВІ ЩІЛЬНОСТІ РОЗПОДІЛУ ПОТОКІВ ДАНИХ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-2

Кудрявський А.О.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Шафроненко А.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кудрявському Артему Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та реалізація методу кластеризіції на основі щільності розподілу потоків даних

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії 03 січня 2025 р.

3. Вихідні дані до роботи науково-технічні публікації щодо дослідження та розробки методу кластеризації на основі потоку даних, вибірки даних з UCI репозиторію.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд основних методів кластеризації потокових даних.
2. Проєктування методу кластеризації на основі щільності потокових даних.
3. Проєктування алгоритму методу кластеризації потокових даних.
4. Визначення математичної моделі проектованого методу.
5. Дослідження реалізованого методу кластеризації потокових даних.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми обробки зображень, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	25.11.24-27.11.24	
3	Аналіз літератури з досліджуваної проблеми	27.11.24-28.11.24	
4	Аналіз технічних засобів	28.11.24-30.11.24	
5	Розробка методу	01.12.24-03.12.24	
6	Програмна реалізація	03.12.24-04.12.24	
7	Оформлення пояснювальної записки	04.12.24-05.12.24	
8	Перевірка на плагіат	06.12.2024	
9	Рецензування	10.12.2024	
10	Підготовка презентації та доповіді	10.12.24-15.12.24	
11	Занесення роботи в електронний архів	01.01.2025	
12	Попередній захист кваліфікаційної роботи	13.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Шафроненко А.Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 77 с., 3 табл., 25 рис., 4 дод., 38 джерел.

КЛАСТЕРИЗАЦІЯ ПОТОКОВИХ ДАНИХ, ГРИДОВА СТРУКТУРА, СТАРІННЯ ДАНИХ, АДАПТАЦІЯ ПАРАМЕТРІВ, ВИЯВЛЕННЯ АНОМАЛІЙ, МЕТОДИ НА ОСНОВІ ЩІЛЬНОСТІ.

Об’єкт дослідження – процес кластеризації потокових даних із використанням методів на основі щільності.

Мета дослідження – розробка методу кластеризації потокових даних для обробки в реальному часі, адаптації до змін структури даних, виявлення кластерів довільної форми та обробки шуму.

Проведено аналіз методів DBSCAN, OPTICS, DenStream, CluStream, D-Stream, розроблено метод кластеризації на основі гридової структури, старіння даних та адаптації параметрів. Реалізовано програмне забезпечення, протестовано на наборах даних різної складності, проаналізовано ефективність.

Результати демонструють високу продуктивність, стійкість до шуму, адаптивність до змін потоку даних. Система може застосовуватися для моніторингу трафіку, виявлення аномалій у фінансових транзакціях, аналізу сенсорних мереж та обробки великих потоків даних у реальному часі.

DENSITY-BASED CLUSTERING, DATA STREAMS, GRID STRUCTURE, DATA DECAY, PARAMETER ADAPTATION, ANOMALY DETECTION.

The object of the research is the process of clustering data streams using density-based methods.

The aim is to develop a clustering method for real-time processing, adapting to data structure changes, detecting arbitrary-shaped clusters, and handling noise.

A review of DBSCAN, OPTICS, DenStream, CluStream, and D-Stream was conducted, and a clustering algorithm was developed using grid structures, data decay, and parameter adaptation. The software was implemented, tested on datasets of varying complexity, and its efficiency analyzed.

Results show high performance, noise resistance, and adaptability to changing data streams. The system is applicable for traffic monitoring, anomaly detection in financial transactions, sensor network analysis, and large-scale data stream processing in real time.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence (штучний інтелект)

ML – Machine Learning (машинне навчання)

NLP – Natural Language Processing (обробка природної мови)

DBSCAN – Density-Based Spatial Clustering of Applications with Noise (алгоритм кластеризації на основі щільності)

OPTICS – Ordering Points To Identify the Clustering Structure (алгоритм кластеризації на основі впорядкування точок)

D-Stream – Density Stream Clustering (метод кластеризації потокових даних)

DenStream – Density-Based Stream Clustering (алгоритм кластеризації потокових даних із механізмом старіння)

CluStream – Clustering Evolving Data Streams (метод кластеризації еволюціонуючих потокових даних)

Grid-based clustering – кластеризація на основі гридової структури

Data decay – старіння даних, механізм для зменшення ваги застарілої інформації

Anomaly detection – виявлення аномалій у потокових даних

Cluster – кластер, група об’єктів із подібними характеристиками

Noise data – шумові дані, які не належать жодному кластеру

Grid cell – грид-клітинка, елементарний сегмент у гридовій структурі

Density threshold – поріг щільності, мінімальна кількість точок для активності гриду

Real-time processing – обробка даних у реальному часі

Feature vector – вектор ознак, числове представлення характеристик об’єкта

Sensor network – сенсорна мережа, система датчиків для збору та аналізу даних

Big Data – великі дані, масиви інформації, що потребують спеціальних методів обробки

ВСТУП

У сучасному світі швидкий прогрес інформаційних технологій спричинюють значне зростання обсягу даних від різних джерел – сенсорів у реальному часі. Ця постійна та велика кількість даних відома як потокова інформація. Обробка таких даних у режимі реального часу стають вагомим завданням у багатьох галузях застосування, наприклад кібербезпека, моніторинг навколишнього середовища, аналіз трафіку та обробка фінансової інформації.

З огляду на швидке зростання обсягів даних та зростаючу потребу в їх оперативному аналізі, розробка ефективних методів кластеризації потокових даних є надзвичайно актуальною. Потокові дані часто містять цінну інформацію, яка може бути використана для прийняття важливих рішень у реальному часі. Наприклад, виявлення аномалій у мережевому трафіку може допомогти запобігти кіберзагрозам, а аналіз даних з сенсорних мереж – оптимізувати управління міською інфраструктурою. Традиційні методи кластеризації не здатні впоратися з динамічністю та великомасштабністю потокових даних, тому потрібні спеціальні алгоритми, які можуть обробляти дані ефективно та вчасно. Крім того, стрімка динаміка потокових даних часто супроводжується зміною розподілу даних, що потребує адаптивних підходів для підтримки актуальності кластерів. Таким чином, удосконалення методів кластеризації на основі щільності є важливим науковим завданням, здатним зробити значний внесок у розвиток технологій обробки великих даних.

Актуальність дослідження полягає у можливості застосування розробленої системи для аналізу потокових даних у реальному часі в різних галузях, таких як економіка, інформаційні технології, медицина, банківська сфера. Система може бути корисною для моніторингу мережевого трафіку, аналізу фінансових операцій, роботи із сенсорними мережами, а також для виявлення аномалій у великих інформаційних потоках.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ КЛАСТЕРИЗАЦІЇ ПОТОКОВИХ ДАНИХ

1.1 Огляд сучасних методів кластеризації потокових даних

З кожним роком обсяги даних, які генеруються в режимі реального часу, стрімко зростають. Потокові дані надходять із різноманітних джерел, таких як сенсорні мережі, системи кібербезпеки, фінансові сервіси, соціальні мережі та інші. У таких умовах традиційні методи кластеризації часто виявляються неефективними через їхню обмеженість у роботі з великими обсягами динамічних даних [1].

Одним із найбільш перспективних напрямків у цій галузі є методи кластеризації на основі щільності. Такі підходи дозволяють виявляти кластери довільної форми, ігнорувати шумові дані та адаптуватися до складних структур. Проте їхнє застосування до потокових даних вимагає вирішення таких викликів:

– динамічність даних – потокові дані постійно оновлюються, що потребує швидких і адаптивних алгоритмів кластеризації;

– обмежені ресурси – великі обсяги даних часто неможливо зберігати повністю, тому необхідно використовувати підходи, які працюють із компактними представленнями;

– швидкість обробки – для забезпечення роботи в реальному часі алгоритми повинні мати низьку обчислювальну складність [2].

На сьогодні існує низка алгоритмів кластеризації, які враховують ці аспекти. Зокрема, методи, засновані на щільності, такі як DBSCAN і OPTICS, були адаптовані для роботи з потоковими даними. Інші підходи, такі як DenStream, CluStream і D-Stream, розроблені спеціально для роботи з динамічними потоками даних.

Незважаючи на досягнення в цій галузі, залишаються актуальними такі питання:

– як забезпечити високу точність кластеризації без значного збільшення обчислювальних витрат?

– як адаптувати методи до зміни розподілу даних у потоках?

– як ефективно використовувати пам’ять під час обробки великих обсягів потокових даних?

Таким чином, удосконалення існуючих підходів і розробка нових методів кластеризації потокових даних є важливим завданням для сучасних дослідників. У подальших розділах роботи буде проведено детальний аналіз найбільш відомих методів кластеризації потокових даних, зокрема DBSCAN, OPTICS, DenStream, CluStream і D-Stream, що дозволить обґрунтувати вибір найбільш ефективного підходу для вирішення поставлених задач [3].

1.2 Метод DBSCAN (Density-Based Spatial Clustering of Applications with Noise

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) є одним із найпоширеніших методів кластеризації, заснованих на щільності. Цей алгоритм дозволяє знаходити кластери довільної форми в наборах даних, які можуть містити значну кількість шуму. Основна ідея методу полягає у виявленні щільних областей даних, відокремлених зонами з низькою щільністю.

Алгоритм (рис. 1.1) базується на двох ключових параметрах: радіусі ϵ, який визначає відстань між сусідніми точками, та мінімальній кількості точок *MinPts*, що необхідна для формування кластера. Якщо точка має достатньо сусідів у межах радіуса ϵ, вона вважається ядровою. Кластери формуються шляхом об’єднання таких ядрових точок та їхніх сусідів. Цей підхід забезпечує

автоматичне визначення кількості кластерів без необхідності попереднього завдання їхньої кількості.

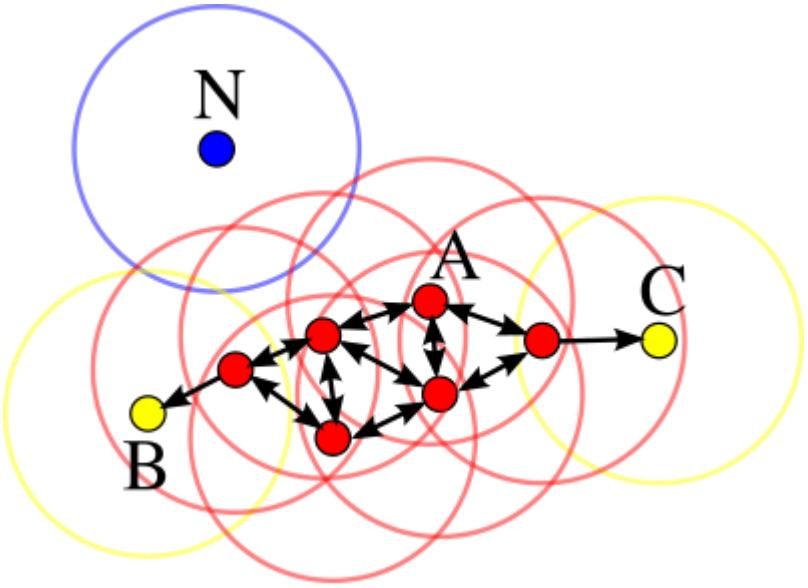

Рисунок 1.1 – Принцип роботи алгоритму DBSCAN

Алгоритм працює за наступними основними кроками:

Крок 1. Для кожної точки p у наборі даних D визначається її ϵ-оточення за формулою:

$$N\epsilon(p) = \{q \in D \mid dist(p,q) \leq \epsilon\}, \quad (1.1)$$

де $dist(p,q)$ – метрика відстані між точками p і q. Найчастіше використовується евклідова відстань:

$$dist(p,q) = \sqrt{(\sum(i = 1)\text{^}d\ (pi - qi)2)}, \quad (1.2)$$

де p_i та q_i – координати точок у i-й вимірності.

Крок 2. На основі кількості точок у є\epsilonє-оточенні визначається, чи є точка p ядровою. Якщо виконується:

$$| N\epsilon(p) | \geq MinPts \quad (1.3)$$

точка класифікується як ядрова. Якщо точка знаходиться в ϵ-оточенні ядрової точки, але сама не має достатньої кількості сусідів, вона вважається прикордонною. Усі інші точки класифікуються як шумові.

Крок 3. Починаючи з ядрових точок, формуються кластери. Алгоритм додає до кластеру всі точки, досяжні з ядрової точки:

$$C = \{q \in D \mid q \text{ досяжна з } p, \text{ де } p \text{ є ядровою точкою}\}. \quad (1.4)$$

Прикордонні точки також включаються до кластеру, якщо вони належать до ϵ-оточення ядрової точки. Цей процес повторюється, доки всі точки, досяжні з ядрової точки, не будуть оброблені.

Крок 4. Після завершення формування одного кластера алгоритм переходить до наступної нерозглянутої точки і повторює процес, доки всі точки набору даних не будуть класифіковані.

Для роботи з даними DBSCAN використовує матрицю відстаней:

$$M[i,j] = dist(pi, pj), \quad (1.5)$$

де $M[i,j]$ – відстань між точками p_i та p_j. На основі цієї матриці визначаються є\epsilonє-оточення кожної точки. Наприклад, для набору точок у двовимірному просторі:

$$D = \{(1,2), (2,2), (3,4), (8,8)\}, \quad (1.6)$$

матриця відстаней виглядатиме так:

$$M = \begin{bmatrix} 0 & 1 & 2,83 & 9,22 \\ 1 & 0 & 2,24 & 8,49 \\ 2,83 & 2,24 & 0 & 6,40 \\ 9,22 & 8,49 & 6,40 & 0 \end{bmatrix}. \quad (1.7)$$

На основі цієї матриці та параметрів $\epsilon = 3$ і $MinPts = 2$ визначаються ядрові точки та формуються кластери.

Цей алгоритм ефективно виявляє кластери довільної форми, що робить його універсальним у багатьох сферах. Наприклад, у геоінформаційних системах DBSCAN використовується для аналізу просторових даних, таких як виявлення скупчень будівель або природних об'єктів. У сфері кібербезпеки алгоритм застосовується для аналізу мережевого трафіку та виявлення аномалій. Він також широко використовується в біоінформатиці для кластеризації біологічних даних, таких як гени або білки.

Попри свої численні переваги, DBSCAN має і певні обмеження. Одним із ключових недоліків є його чутливість до вибору параметрів ϵ і *MinPts*. Якщо радіус ϵ обрано неправильно, кластеризація може виявитися неефективною. Наприклад, якщо радіус занадто малий, кластери можуть бути розбиті на кілька дрібних груп, тоді як занадто великий радіус може призвести до об'єднання кількох кластерів в один. Крім того, алгоритм погано масштабується на великі набори даних через квадратичну обчислювальну складність $O(n^2)$, що обмежує його застосування в умовах, коли дані надходять у реальному часі [4].

1.3 Метод OPTICS (Ordering Points To Identify the Clustering Structure)

OPTICS (Ordering Points To Identify the Clustering Structure) – це алгоритм кластеризації, який розширює ідеї DBSCAN і дозволяє виявляти кластери з різною щільністю. Основна ідея методу полягає у впорядкуванні точок набору даних таким чином, щоб було видно, як змінюється щільність

між ними. Це дає змогу уникнути необхідності задавати фіксоване значення параметра ϵ, яке є обмеженням у DBSCAN, і дозволяє працювати з даними, де щільність варіюється.

На відміну від DBSCAN, OPTICS (рис. 1.2) не виконує кластеризацію безпосередньо. Натомість він створює впорядковану послідовність точок із записаними відстанями досяжності (reachabilitydistances) які відображають, наскільки тісно пов'язані точки одна з одною. Результатом роботи алгоритму є «плот відстаней досяжності» (reachability plot), на якому локальні мінімуми відповідають початкам кластерів.

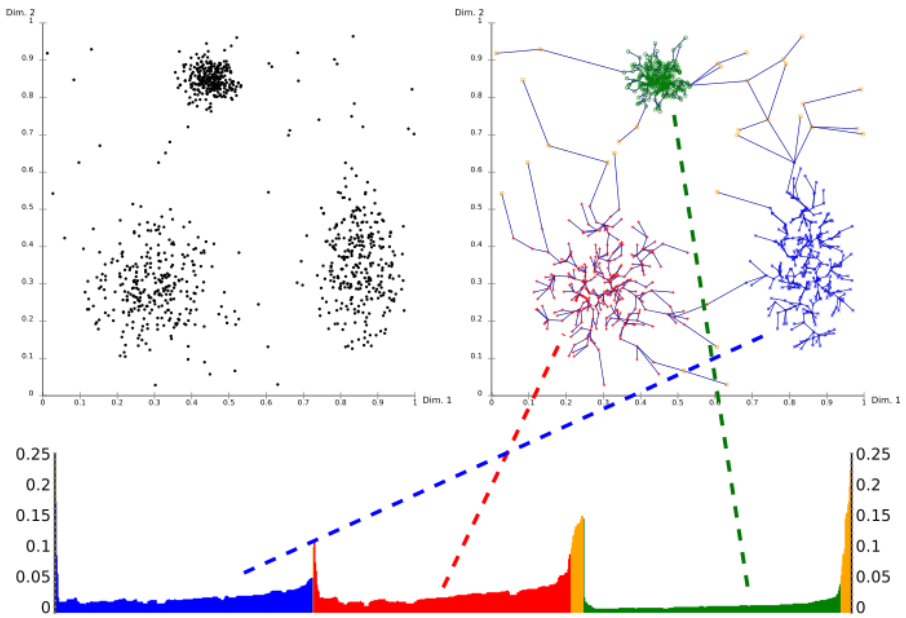

Рисунок 1.2 – Принцип роботи алгоритму OPTICS

Алгоритм використовує концепції відстані ядра (coredistancecore distancecoredistance) та відстані досяжності (reachabilitydistancereachability distancereachabilitydistance), які визначають, наскільки щільно розташовані точки навколо заданої точки. Відстань ядра характеризує щільність сусідніх точок, тоді як відстань досяжності показує, наскільки близько розташовані інші точки щодо ядра.

OPTICS працює за наступними основними кроками:

Крок 1. Для кожної точки p у наборі даних визначається відстань ядра (*core_dist*(p)). Якщо точка має достатню кількість сусідів (визначену параметром *MinPts*) у межах радіуса ϵ, її відстань ядра дорівнює відстані до її *MinPts* - го найближчого сусіда:

$$core\_distance(p) = \{distance(p, MinPts - th\ nearest\ point), if\ |N_{\varepsilon(p)}| \geq MinPts\ undefined, otherwise\}. \quad (1.8)$$

Крок 2. Алгоритм починає з довільної точки p і визначає всі точки в її ϵ-оточенні. Для кожної сусідньої точки q обчислюється відстань досяжності:

$$reachability\_dist(p, q) = max(core\_dist(p), dist(p, q)). \quad (1.9)$$

Точки сортуються за зростанням відстані досяжності та додаються до впорядкованої послідовності.

Крок 3. Процес повторюється для кожної нерозглянутої точки набору даних. Якщо точка не має ядрової відстані (*core_dist* = ∞) вона вважається шумовою або прикордонною.

Крок 4. На основі отриманої послідовності будується графік відстаней досяжності. Кластери визначаються як області з локальними мінімумами на цьому графіку.

Для роботи алгоритму використовується матриця відстаней, подібно до DBSCAN:

$$M[i, j] = dist(pi, pj), \quad (1.10)$$

де $M[i,j]$ – відстань між точками p_i та p_j. Матриця є основою для обчислення ϵ-оточення кожної точки та побудови впорядкованої послідовності.

Наприклад, для набору точок D = {(1,2),(2,2),(3,3),(8,8)} і параметрів ϵ = 3, *MinPts* = 2, матриця виглядає так:

$$M = \begin{bmatrix} 0 & 1 & 2,24 & 9,42 \\ 1 & 0 & 1,41 & 8,49 \\ 2,24 & 1,41 & 0 & 7,07 \\ 9,22 & 8,49 & 7,07 & 0 \end{bmatrix}. \quad (1.11)$$

На основі цієї матриці обчислюються core_dist та reachability_dist для кожної точки, а потім будується послідовність, яка використовується для виявлення кластерів.

Основною перевагою OPTICS є його здатність виявляти кластери змінної щільності, що робить його більш універсальним у порівнянні з DBSCAN. Крім того, алгоритм дає можливість візуалізувати щільність даних за допомогою графіка відстаней досяжності. Однак він має обмеження, подібні до DBSCAN, зокрема високу обчислювальну складність ($O(n^2)$) та залежність від параметра *MinPts*.

Попри свої недоліки, OPTICS широко використовується в задачах аналізу складних наборів даних, особливо у випадках, коли щільність кластерів значно варіюється. Це робить його ефективним для аналізу просторових даних, виявлення аномалій у мережевому трафіку та інших задач, де структура даних є складною [5].

1.4 Метод DenStream

DenStream (Density-Based Stream Clustering) є одним із найбільш популярних алгоритмів для кластеризації потокових даних. Його основною метою є забезпечити обробку даних, які надходять у реальному часі, з урахуванням обмежень пам’яті та швидкості обчислень. DenStream (рис. 1.3) розроблений для роботи в умовах, де дані постійно оновлюються, а їхня структура може змінюватися. Це робить його надзвичайно корисним у таких

сферах, як моніторинг мережевого трафіку, аналіз сенсорних мереж і обробка фінансових транзакцій [6].

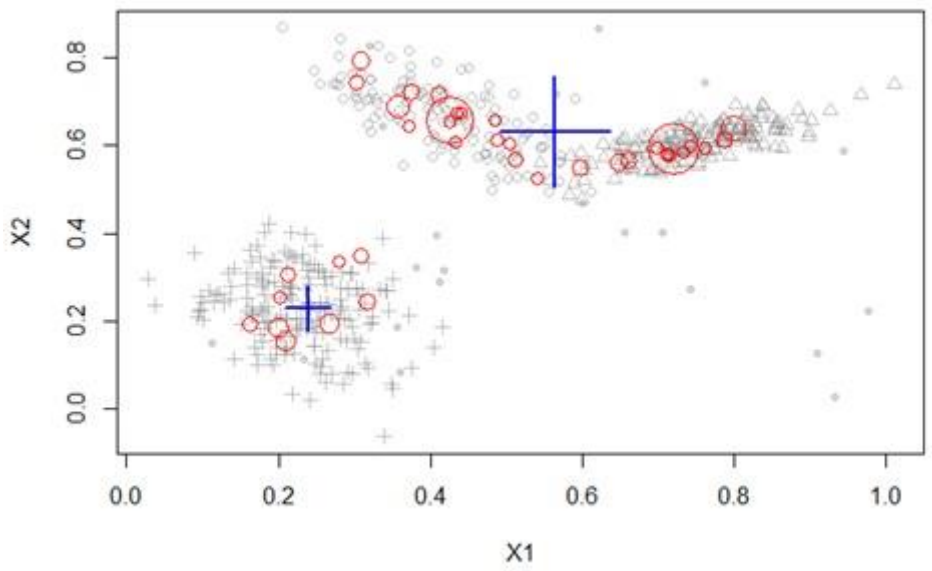

Рисунок 1.3 – Принцип роботи алгоритму DenStream

Метод заснований на ідеї підтримки компактного представлення даних за допомогою мікрокластерів. Мікрокластери – це компактні об’єкти, які узагальнюють інформацію про невеликі групи точок, зберігаючи при цьому їхню основну структуру. DenStream виділяє два типи мікрокластерів: потенційні (*pMC*) і викинуті (*oMC*). Потенційні мікрокластери представляють області з високою щільністю точок, які можуть стати стабільними кластерами. Викинуті мікрокластери відповідають областям із низькою щільністю, які найчастіше розглядаються як шум або тимчасові об’єкти.

Однією з важливих особливостей DenStream є використання часової ваги для врахування старіння точок у потоці. Це означає, що дані, які надходили раніше, поступово втрачають свою важливість, а більша увага приділяється актуальним даним. Такий підхід дозволяє алгоритму адаптуватися до змін у структурі потокових даних.

DenStream працює за наступними основними кроками:

Крок 1. Ініціалізація – алгоритм починає з формування мікрокластерів на основі первинного набору даних. Для цього використовується поріг ϵ\epsilonϵ для визначення ϵ - оточення точок і параметр *MinPts* для класифікації точок як ядрових.

Крок 2. Додавання нових точок – для кожної нової точки у потоці визначається її відстань до існуючих мікрокластерів. Якщо точка належить до ϵ-оточення одного з мікрокластерів, цей мікрокластер оновлюється. Якщо точка не належить до жодного мікрокластера, створюється новий потенційний або викинутий мікрокластер.

Крок 3. Оновлення мікрокластерів – усі мікрокластери періодично оновлюються із застосуванням часової ваги. Це дозволяє враховувати старіння точок у потоці. Часова вага обчислюється за формулою:

$$w(t) = e - \lambda(T - t), \tag{1.12}$$

де T – поточний час;

t – час появи точки;

λ – параметр, що визначає швидкість старіння.

Крок 4. Фільтрація викинутих мікрокластерів – мікрокластери, чия щільність опускається нижче порогового значення, видаляються.

Крок 5. Формування кінцевих кластерів – потенційні мікрокластери об’єднуються в стабільні кластери, якщо їхня щільність залишається високою протягом тривалого часу.

Математично мікрокластери можна описати через їхні характеристики:

Центр мікрокластера:

$$c = \frac{\sum_{i-1}^{n} wi \cdot pi}{\sum_{i-1}^{n} wi}, \tag{1.13}$$

де p_i – координати точки;

w_i – її вага.

Радіус мікрокластера:

$$r = \sqrt{\frac{\sum_{i-1}^{n} wi \cdot dist(pi,c)}{\sum_{i-1}^{n} wi}}, \tag{1.14}$$

де $dist(pi,c)$ – відстань між точкою p_i і центром c.

Наприклад, для потоку даних, що надходить у вигляді послідовності точок $D = \{(1,2),(2,3),(3,4),\dots\}$, мікрокластери можуть виглядати так:

$$pMC = \{(2.5,3.5, w = 5), (10,12, w = 3)\}, \tag{1.15}$$

де кожен мікрокластер характеризується центром, радіусом і загальною вагою.

DenStream є потужним інструментом для кластеризації потокових даних завдяки своїй здатності динамічно адаптуватися до змін у потоці. Основною перевагою методу є використання часової ваги, яка дозволяє алгоритму «забувати» старі точки і зосереджуватися на актуальних даних. Однак алгоритм залежить від параметрів ϵ, *MinPts* і λ, які можуть суттєво впливати на результати кластеризації.

Незважаючи на свої обмеження, DenStream широко використовується у завданнях моніторингу мережевого трафіку, аналізу сенсорних даних, а також у системах виявлення шахрайства, де необхідна обробка великих обсягів даних у реальному часі [7].

1.5 Метод CluStream

CluStream (Clustering Data Streams) – це один із перших алгоритмів, розроблених спеціально для обробки потокових даних. Його основною метою є забезпечення можливості динамічного групування точок, що надходять у реальному часі, із подальшим використанням отриманих кластерів для аналізу

історичних даних. Метод заснований на концепції мікрокластерів, які агрегують інформацію про точки потоку, дозволяючи ефективно працювати з великими обсягами даних [8].

CluStream (рис. 1.4) виконує кластеризацію у два етапи: онлайн і офлайн. На онлайн-етапі алгоритм підтримує актуальний стан мікрокластерів, зберігаючи статистичні дані про поточний потік. На офлайн-етапі виконується аналіз цих мікрокластерів для побудови остаточних кластерів залежно від конкретних вимог або цілей аналізу.

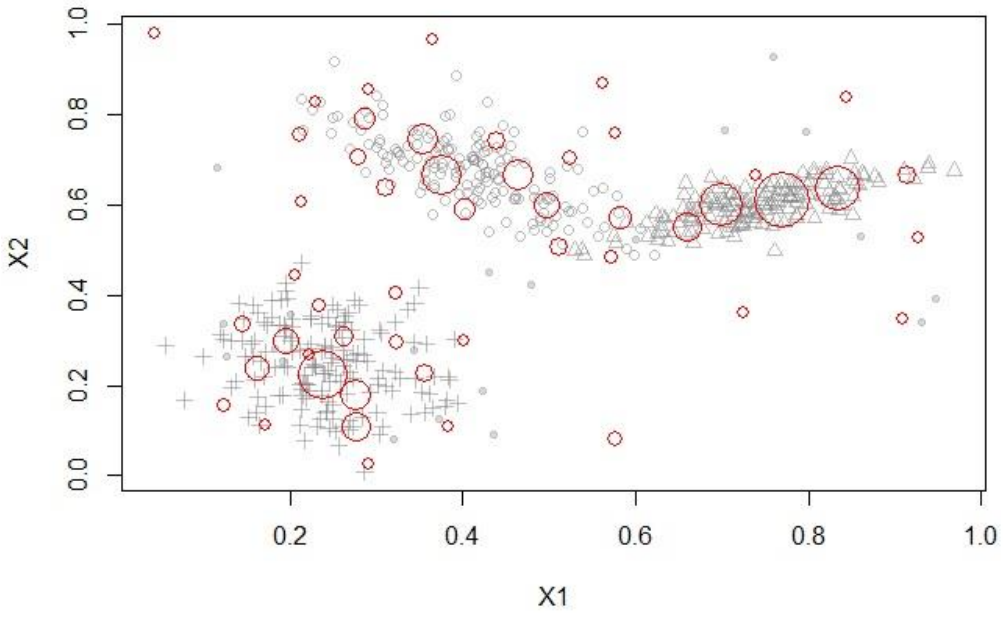

Рисунок 1.4 – Принцип роботи алгоритму CluStream

Мікрокластери в CluStream характеризуються компактним представленням даних, яке включає:

– суму координат точок (CF_1) для обчислення центру;

– суму квадратів координат (CF_2) для обчислення розсіювання;

– кількість точок у мікрокластері (n).

Ці характеристики дозволяють обчислювати основні статистичні параметри мікрокластера, такі як його центр та радіус, що робить метод ефективним для обробки великих потоків даних.

CluStream працює за наступними основними кроками:

Крок 1. Ініціалізація – алгоритм починається з формування початкових мікрокластерів на основі вибірки історичних даних. Для цього використовується метод *k-means*, щоб розподілити точки на k початкових кластерів.

Крок 2. Оновлення мікрокластерів – при надходженні нової точки перевіряється її належність до одного з існуючих мікрокластерів. Якщо точка потрапляє до області впливу мікрокластера, то статистичні параметри цього мікрокластера оновлюються за формулою:

$$CF1 \leftarrow CF1 + xi, CF2 \leftarrow CF2 + xi2, n \leftarrow n + 1, \quad (1.16)$$

де x_i – координати нової точки.

Крок 3. Створення нового мікрокластера – якщо точка не потрапляє до жодного з існуючих мікрокластерів, створюється новий мікрокластер. У разі досягнення максимальної кількості мікрокластерів відбувається злиття або видалення найменш значущого мікрокластеру.

Крок 4. Забезпечення часової релевантності – CluStream враховує старіння точок, зменшуючи вагу старих даних через часову функцію:

$$w(t) = e - \lambda(T - t), \quad (1.17)$$

де T – поточний час;

t – час появи точки;

λ – параметр, що визначає швидкість старіння.

Крок 5. Офлайн-аналіз – на цьому етапі мікрокластери використовуються для виконання запитів до даних або побудови остаточних кластерів. Для цього застосовуються традиційні методи кластеризації, такі як *k - means* або *DBSCAN*, залежно від цілей аналізу.

Мікрокластери в CluStream можна описати через їхні характеристики.

Центр мікрокластера:

$$\mathrm{c} = \frac{CF1}{n}, \tag{1.18}$$

де CF_1 – сума координат точок;

n – кількість точок у мікрокластері.

Радіус мікрокластера:

$$r = \sqrt{\frac{CF2}{n} - \frac{CF1}{n}}, \tag{1.19}$$

де CF_2 – сума квадратів координат точок.

Наприклад, для потоку даних D ={(1,2),(2,3),(3,4),… }, мікрокластери можуть бути описані так:

$$MC = \{(c = (2,3), r = 1.5, n = 10), \dots\}, \tag{1.20}$$

де кожен мікрокластер характеризується центром, радіусом і кількістю точок.

CluStream має низку переваг, зокрема можливість обробки даних у реальному часі та аналізу історичних даних. Це робить метод надзвичайно корисним у ситуаціях, коли дані надходять безперервно, і важливо підтримувати актуальне представлення кластерів. Його гнучкість дозволяє використовувати мікрокластери для побудови підсумкових кластерів різного рівня деталізації, залежно від потреб аналізу. Крім того, підтримка статистичних характеристик, таких як центр і радіус, забезпечує ефективне виконання запитів до даних із мінімальними обчислювальними витратами.

Проте метод має і свої обмеження. Він чутливий до вибору параметрів k, λ та максимальної кількості мікрокластерів. Неправильний вибір цих параметрів може призвести до втрати важливої інформації або створення занадто великої кількості кластерів, що ускладнює їхній подальший аналіз. Ще

однією проблемою є те, що CluStream не враховує суттєві зміни в потоках даних, які можуть призводити до значного старіння кластерів.

Незважаючи на ці недоліки, CluStream залишається одним із найефективніших методів для кластеризації потокових даних. Його застосовують у багатьох галузях, таких як виявлення шахрайства у фінансових операціях, моніторинг мережевого трафіку та аналіз даних із сенсорних мереж. Завдяки своїй адаптивності та здатності працювати в умовах реального часу CluStream є незамінним інструментом для вирішення сучасних завдань аналізу потокових даних [9].

1.6 Метод D-Stream

D-Stream (Density-Based Clustering for Data Streams) є алгоритмом кластеризації потокових даних, що використовує сіткову (гридову) структуру для представлення простору даних. Основна ідея методу полягає у поділі простору даних на клітинки (гриди), щільність яких використовується для визначення кластерів. Такий підхід дозволяє алгоритму ефективно працювати з великими потоковими даними, підтримуючи їх компактне представлення.

D-Stream (рис. 1.5) виділяється тим, що дозволяє одночасно обробляти як нові, так і історичні дані завдяки використанню моделі часової ваги. Алгоритм динамічно адаптується до змін у потоці даних, автоматично оновлюючи щільність гридів та структуру кластерів.

Метод D-Stream працює у три основні фази: ініціалізація, онлайн-обробка та офлайн-аналіз. На етапі ініціалізації простір даних поділяється на гриди, які характеризуються координатами, розміром і щільністю. У режимі онлайн алгоритм оновлює щільності гридів у реальному часі, враховуючи часову вагу. На етапі офлайн-аналізу щільні гриди групуються в кластери, а рідкі гриди позначаються як шум.

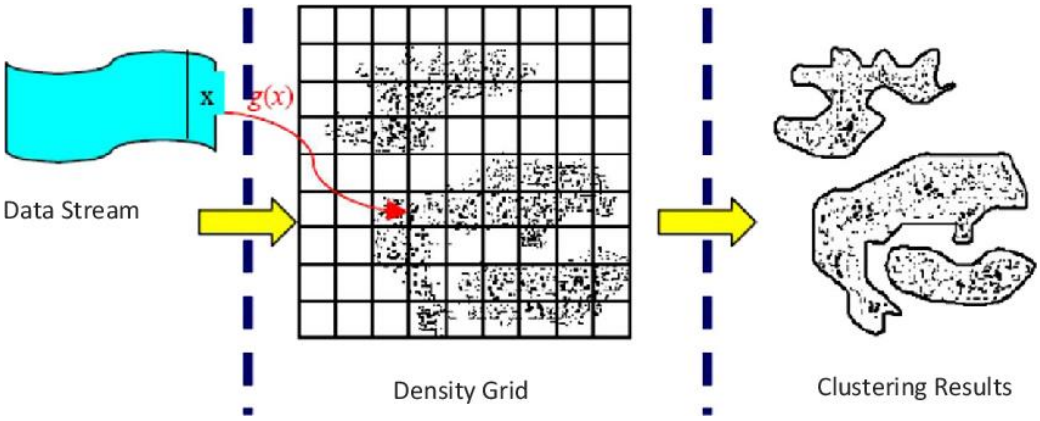

Рисунок 1.5 – Принцип роботи алгоритму D-Stream

D-Stream працює за наступними основними кроками:

Крок 1. Ініціалізація – простір даних ділиться на рівномірні клітинки (гриди) розміром δ. Для кожного гриду визначається його початкова щільність. Грид із високою щільністю позначається як активний, із середньою позначається як потенційний, а з низькою – як рідкий (шум).

Крок 2. Оновлення щільності гридів – при надходженні нової точки її координати використовуються для визначення гриду, до якого вона належить. Щільність цього гриду оновлюється за формулою:

$$D(g,t) = D(g,t-1) \cdot e - \lambda(t - t0) + 1, \qquad (1.21)$$

де $D(g,t)$ – щільність гриду g у момент часу t;

λ – параметр затухання;

t_0 – час останнього оновлення гриду.

Крок 3. Фільтрація неактивних гридів – гриди з дуже низькою щільністю автоматично видаляються для зменшення обчислювальних витрат. Це дозволяє зосередитися на обробці значущих даних.

Крок 4. Формування кластерів – на основі щільності гридів виконується групування активних гридів у кластери. Гриди вважаються пов’язаними, якщо вони є сусідніми (мають спільну грань або кут).

Крок 5. Аналіз у реальному часі – алгоритм регулярно оновлює кластери, враховуючи нові дані та видаляючи застарілі гриди. Це дозволяє підтримувати актуальність отриманих кластерів.

Математично щільність гридів описується як:

$$D(g) = \frac{\text{кількість точок у гриді}}{\text{об}'\text{єм гриду}}. \quad (1.22)$$

Гриди поділяються на активні, потенційні та рідкі залежно від їхньої щільності:

– активні гриди ($D(g) > \theta_a$) формують основні частини кластерів;

– потенційні гриди ($\theta_p \leq D(g) \leq \theta_a$) можуть стати частиною кластерів у майбутньому;

– рідкі гриди ($D(g) < \theta_p$) вважаються шумом і не включаються до кластерів.

Наприклад, для потоку даних у двовимірному просторі $D = \{(1,2),(2,2),(3,3),\dots\}$ при розмірі гриду $\delta = 1$, активні гриди можуть виглядати так:

$$G = \{(1,2): D(g) = 5, (2,2): D(g) = 8, \dots\}, \quad (1.23)$$

для ілюстрації, простір даних у двовимірному просторі може бути представлений у вигляді матриці гридів:

$$\begin{bmatrix} 0 & 0 & 5 & 2 \\ 0 & 10 & 8 & 0 \\ 3 & 7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.24)$$

де кожен елемент матриці відповідає щільності відповідного гриду. Формування кластерів ґрунтується на взаємодії сусідніх гридів. Два гриди вважаються сусідніми, якщо виконуються умови:

$$dist(gi, gj) \leq \delta, \quad (1.25)$$

де $dist(g_i,g_j)$ – відстань між центрами гридів g_i і g_j;

δ – розмір гриду.

D-Stream має кілька важливих переваг. По-перше, метод добре масштабується на великі обсяги даних, оскільки працює із сітковою структурою, яка значно зменшує обсяг обчислень. По-друге, він адаптується до змін у потоці даних, автоматично оновлюючи щільності гридів і кластерну структуру. Однак метод також має свої обмеження, зокрема залежність від вибору параметрів δ, λ і порогів θa,θp. Неправильний вибір цих параметрів може призвести до втрати значущих даних або створення надто великої кількості кластерів.

Незважаючи на ці недоліки, D-Stream є ефективним інструментом для аналізу потокових даних у реальному часі. Його застосовують у задачах моніторингу навколишнього середовища, аналізу сенсорних мереж і виявлення аномалій у великих потоках даних [10].

1.7 Постановка задачі дослідження

З урахуванням вищезазначеного було визначено, що існує потреба у розробці нового підходу до кластеризації потокових даних, який об’єднує переваги методів на основі щільності з можливістю динамічної адаптації до змін у потоках даних. Це дозволить забезпечити високу точність кластеризації, адаптивність і ефективність обчислень.

Об’єкт дослідження – процес кластеризації потокових даних із використанням методів на основі щільності.

Мета дослідження – розробка методу кластеризації потокових даних для обробки в реальному часі, адаптації до змін структури даних, виявлення кластерів довільної форми та обробки шуму.

Для досягнення мети необхідно вирішити такі завдання:

– розробити математичну модель для кластеризації потокових даних, яка враховує щільність, часову вагу та взаємозв’язок точок у потоці;

– створити алгоритм, який працює із сітковою структурою або мікрокластерами для ефективної обробки великих обсягів даних;

– реалізувати програмне забезпечення, яке інтегрує запропонований метод кластеризації;

2 ПРОЄКТУВАННЯ МЕТОДУ КЛАСТЕРИЗАЦІЇ НА ОСНОВІ ЩІЛЬНОСТІ ПОТОКОВИХ ДАНИХ

2.1 Загальна характеристика проектованого методу кластеризації потокових даних

Кластеризація потокових даних є важливою задачею для багатьох галузей, що працюють із великими обсягами інформації, яка постійно оновлюється. Проектований метод спрямований на вирішення проблеми аналізу таких даних у режимі реального часу із забезпеченням високої точності кластеризації та ефективного використання ресурсів. Проектований метод базується на використанні гридової структури для представлення простору даних. Простір ділиться на клітинки (гриди) фіксованого розміру, у кожній із яких обчислюється щільність даних. Такий підхід дозволяє:

– виділяти активні області з високою щільністю даних;

– ігнорувати шумові дані з низькою щільністю;

– формувати кластери довільної форми на основі сусідніх активних гридів.

Метод також передбачає врахування динаміки потокових даних завдяки механізму старіння, який знижує вагу старих даних у часі. Це забезпечує актуальність кластерів та мінімізує вплив застарілої інформації.

Основними елементами проектованого методу є:

– гридова структура – простір ділиться на клітинки, що дозволяє знизити обчислювальну складність при аналізі великих обсягів даних;

– механізм оновлення щільності – щільність кожного гриду змінюється із часом залежно від нових даних і процесу старіння;

– фільтрація шуму – дані з низькою щільністю відкидаються, що дозволяє уникнути впливу нерелевантної інформації;

– формування кластерів – кластери утворюються шляхом об’єднання сусідніх гридів із високою щільністю;

– адаптація параметрів – метод передбачає автоматичне налаштування параметрів, таких як розмір гриду та пороги щільності, залежно від особливостей даних.

Проектований метод орієнтований на застосування в таких галузях, як аналіз мережевого трафіку, виявлення аномалій у фінансових операціях, обробка даних сенсорних мереж тощо. Його особливістю є здатність працювати в реальному часі, забезпечуючи високу точність та адаптивність. Це робить метод придатним для вирішення задач, де важливими є швидкість прийняття рішень та актуальність результатів. Наприклад, виявлення кіберзагроз у реальному часі вимагає негайної класифікації даних, а аналіз сенсорних мереж може допомогти в оптимізації ресурсів у смарт-системах, таких як розумні міста чи промислові IoT-системи. Гнучкість і адаптивність методу також відкривають перспективи для його використання у сферах прогнозування, таких як моніторинг ринку або управління логістичними процесами.

2.2 Проєктування алгоритму роботи методу кластеризації потокових даних

Розробка ефективного алгоритму кластеризації потокових даних є ключовим завданням для аналізу великих обсягів інформації, яка постійно змінюється. Основна ідея алгоритму полягає в тому, щоб забезпечити обробку даних у реальному часі, виділяючи кластери на основі щільності даних у просторі. При цьому важливо враховувати специфіку потоків: постійне надходження нових точок, старіння старих даних та наявність шуму. Одним із головних викликів при проектуванні алгоритму є необхідність досягнення балансу між точністю кластеризації та швидкістю роботи. Потокові дані

характеризуються великими обсягами та динамічною структурою, тому метод має бути адаптивним, щоб своєчасно реагувати на зміни в даних і забезпечувати формування релевантних кластерів. Використання гридової структури дозволяє суттєво знизити обчислювальні витрати, зберігаючи при цьому високу якість аналізу.

Алгоритм включає кілька ключових етапів, кожен із яких спрямований на вирішення конкретного аспекту обробки даних: ідентифікація областей високої щільності, фільтрація нерелевантних даних, формування кластерів і динамічна адаптація параметрів. Ці етапи взаємопов’язані та забезпечують цілісну роботу алгоритму, орієнтовану на отримання максимально точних результатів у реальних умовах. Наведемо ці етапи:

Етап 1. «Ідентифікація гридів» – першим етапом алгоритму є поділ простору на клітинки (гриди) фіксованого розміру. Для кожної точки обчислюється унікальний ідентифікатор гриду, до якого вона належить. Це дозволяє швидко класифікувати точки, зменшуючи обчислювальну складність навіть для великих обсягів даних. Вибір розміру гридів (grid_size) визначає точність кластеризації: менший розмір забезпечує високу деталізацію, але збільшує кількість гридів.

Етап 2. «Оновлення щільності гридів» – щільність кожного гриду оновлюється із врахуванням двох факторів: кількості нових точок, які потрапили до гриду, та старіння даних. Щільність знижується експоненційно, якщо в грид не надходять нові точки, що дозволяє ігнорувати застарілі дані. Це є ключовим елементом, який забезпечує актуальність інформації у динамічних потоках даних.

Етап 3. «Фільтрація неактивних гридів» – гриди, щільність яких опускається нижче порогового значення (potential_threshold), вважаються неактивними та видаляються із структури. Це дозволяє значно знизити обсяг пам’яті, необхідної для роботи алгоритму, і зосередитися на обробці лише релевантних областей даних.

Етап 4. «Формування кластерів» – кластери утворюються шляхом об’єднання сусідніх активних гридів із високою щільністю. Алгоритм використовує метод глибокого обходу (DFS) для пошуку всіх пов’язаних гридів, що відповідають критеріям щільності. Це забезпечує можливість формування кластерів довільної форми, що важливо для аналізу реальних даних із нерегулярною структурою.

Етап 5. «Адаптація параметрів» – для збереження стабільної продуктивності алгоритму в умовах змін у структурі даних параметри кластеризації автоматично коригуються. Наприклад, середня щільність гридів може використовуватися для збільшення або зменшення розміру клітинки (grid_size). Такий підхід робить алгоритм більш універсальним та адаптивним. Блок – схему проектованого алгоритму роботи методу кластеризації потокових даних наведено на рисунку 2.1.

Рисунок 2.1 – Блок – схема проектованого алгоритму роботи методу кластеризації потокових даних

У таблиці 2.1 наведено підсумок ключових етапів алгоритму та їхню функціональну роль.

Таблиця 2.1 – Основні етапи алгоритму кластеризації потокових даних

№	**Етап алгоритму**	**Опис**	**Функціональна роль**
1.	Ідентифікація гридів	Визначення клітинки, до якої належить кожна точка.	Оптимізація обчислювальної складності
2.	Оновлення щільності	Розрахунок щільності з урахуванням старіння даних.	Збереження актуальності даних
3.	Фільтрація гридів	Видалення гридів із низькою щільністю.	Зменшення обсягу пам’яті
4.	Формування кластерів	Об’єднання сусідніх активних гридів у цілісні кластери.	Виділення структур із високою щільністю
5.	Адаптація параметрів	Динамічне налаштування параметрів алгоритму залежно від особливостей даних.	Підтримка ефективності в умовах змін

Реалізована структура алгоритму дозволяє ефективно виконувати кластеризацію потокових даних у режимі реального часу, забезпечуючи точність аналізу навіть у складних умовах.

2.3 Визначення математичної моделі проектованого методу

Для забезпечення чіткої основи роботи алгоритму кластеризації потокових даних необхідно сформувати математичну модель, яка описує основні аспекти його функціонування. Ця модель включає опис процесів обчислення щільності гридів, старіння даних, об’єднання активних гридів у кластери та адаптації параметрів для динамічної роботи з потоками даних.

2.3.1 Формули для щільності та старіння гридів

На першому етапі моделі опишемо процеси обчислення щільності гридів та їх старіння з часом. Щільність є ключовим параметром, який визначає активність гриду та його участь у формуванні кластерів. Механізм старіння дозволяє алгоритму враховувати лише актуальні дані, що є особливо важливим у потокових даних, які постійно змінюються. Щільність гриду оновлюється на кожному кроці з урахуванням нових даних та механізму старіння за формулою:

$$Dt(G) = Dt - 1(G) \cdot e - \lambda \Delta t + Nnew(G), \qquad (2.1)$$

де $D_t(G)$ – поточна щільність гриду G у момент часу t;

$D_{t-1}(G)$ – щільність гриду на попередньому кроці;

λ – коефіцієнт старіння;

Δt – проміжок часу між оновленнями;

$N_{new}(G)$ – кількість нових точок, що потрапили до гриду G за останній проміжок часу.

Коефіцієнт старіння λ визначає швидкість, із якою зменшується вагомість старих даних. Наприклад, якщо λ велике, дані швидко втрачають свою актуальність.

Для визначення $N_{new}(G)$ використовується формула:

$$N_{new}(G) = \sum_{i=1}^{n} \delta(Gi, G), \quad (2.2)$$

де n – кількість точок, доданих за час Δt;

$\delta(G_i,G)$ – індикаторна функція, що дорівнює 1, якщо точка G_i належить гриду G, і 0 в іншому випадку.

2.3.2 Модель об’єднання гридів у кластери

Наступним кроком проведемо опис процесу об’єднання гридів у кластери. Кластери формуються шляхом ідентифікації активних гридів і визначення їх сусідства. Це дозволяє створювати структури довільної форми, які точно відображають характеристики даних. Модель об’єднання базується на геометричних та щільнісних критеріях, що забезпечує гнучкість та ефективність. Грид G вважається активним, якщо:

$$D(G) \geq threshold_{active}\ , \quad (2.3)$$

де $threshold_{active}$ – поріг щільності для активного гриду.

Сусідство між гридом G_1 і G_2 визначається умовою:

$$dist(G1, G2) \leq grid\_size, \quad (2.4)$$

де $dist(G_1,G_2)$ – евклідова відстань між центрами гридів;

grid_size – розмір клітинки.

Для формування кластерів застосовується метод глибокого обходу (DFS), який шукає всі активні гриди, пов’язані між собою через сусідів. Якщо $G_1, G_2, \ldots, G_k$ – множина активних гридів, що задовольняє умови:

– $D(G_i) \geq threshold_{active}$;

– $\forall_i, j : dist(G_i, G_j) \leq grid\_size$,

то всі ці гриди об’єднуються в один кластер.

2.3.3 Механізм адаптації параметрів

Для забезпечення стабільності та точності роботи алгоритму в умовах змінної структури даних необхідна адаптація його параметрів. Представимо математичний опис механізмів адаптації, таких як коригування розміру гридів і порогів щільності. Адаптація дозволяє алгоритму підтримувати ефективність незалежно від змін у потокових даних. Основні параметри, що піддаються адаптації:

– розмір гриду (*grid_size*);

– пороги щільності ($threshold_{active}$, $threshold_{potential}$).

Розмір гриду змінюється залежно від середньої щільності гридів у потоці. Новий розмір обчислюється за формулою:

$$grid\_size_{new} = grid\_size_{current} \cdot \sqrt{\frac{D_{current}}{D_{target}}}, \quad (2.5)$$

де $D_{current}$ – середня щільність гридів на поточному етапі;

D_{target} – цільова середня щільність.

Пороги щільності також коригуються на основі статистичних характеристик даних. Активний поріг може бути змінений за формулою:

$$threshhold_{active\_new} = D + \alpha \cdot \sigma D, \quad (2.6)$$

де D – середнє значення щільності;

σD – стандартне відхилення щільності;

α – коефіцієнт, що визначає вплив варіації даних.

Математична модель проектованого алгоритму кластеризації потокових даних забезпечує формалізований опис його роботи. Формули для щільності та старіння гридів дозволяють зберігати актуальність даних у реальному часі. Модель об’єднання гридів визначає правила формування кластерів, а механізм адаптації параметрів забезпечує гнучкість алгоритму в умовах змінної структури даних. Ця модель слугує основою для ефективної реалізації алгоритму в різноманітних сценаріях.

2.4 Порівняння проектованого методу з існуючими методами кластеризації

Наступним кроком для оцінки ефективності та адаптивності проектованого методу кластеризації потокових даних є проведення порівняння з існуючими популярними методами, такими як DBSCAN, OPTICS, DenStream, CluStream та D-Stream. Це порівняння дозволяє виявити переваги та недоліки проектованого підходу, а також визначити, наскільки він відповідає сучасним вимогам до аналізу потокових даних. Для проведення порівняння буде використано набір критеріїв, що охоплюють різні аспекти роботи алгоритмів. Вибрані критерії відображають основні вимоги до кластеризації потокових даних, такі як здатність працювати в реальному часі, адаптація до змін, обчислювальна ефективність, а також можливість виявлення складних структур даних. Додатково буде розглянуто такі критерії, як стійкість до шуму, підтримка обробки великих обсягів даних, спроможність працювати з високовимірними даними та простота налаштування.

Основні критерії порівняння:

– робота з потоковими даними – чи може метод обробляти дані в реальному часі;

– адаптація до змін – здатність алгоритму динамічно змінювати свої параметри відповідно до змін у даних;

– обчислювальна ефективність – рівень складності алгоритму та використання ресурсів;

– виявлення кластерів довільної форми – можливість роботи з нерегулярними та складними структурами;

– стійкість до шуму – здатність ігнорувати нерелевантні дані, які можуть спотворювати результати кластеризації;

– масштабованість – спроможність алгоритму працювати з великими наборами даних без значного зниження продуктивності;

– робота з високовимірними даними – чи ефективно алгоритм обробляє дані з великою кількістю ознак;

– простота налаштування – легкість налаштування параметрів алгоритму для різних сценаріїв.

У таблиці 2.2 наведено порівняння проектованого методу з іншими розглянутими методами кластеризації на основі зазначених критеріїв.

З результатів проведеного порівняльного аналізу можна побачити, що проектований метод кластеризації потокових даних має низку суттєвих переваг у порівнянні з іншими популярними підходами. Однією з ключових особливостей є здатність працювати з потоковими даними в реальному часі, що недоступно для методів DBSCAN і OPTICS. Завдяки цьому проектований метод ідеально підходить для задач, які вимагають постійного оновлення кластерів і обробки великих обсягів інформації.

Ще одним важливим аспектом є адаптація до змін у даних. Використання механізму старіння та динамічного налаштування параметрів дозволяє проектованому методу швидко реагувати на зміну структури потоків,

що є вагомою перевагою перед статичними підходами. Це забезпечує стабільність і актуальність отриманих результатів у динамічних умовах.

Таблиця 2.2 – Порівняльна характеристика проектованого методу з існуючими

Критерій	**Проєктований метод**	**DBSCAN**	**OPTICS**	**Den Stream**	**Clu Stream**	**D - Stream**
Робота з потоковими даними	Так	Ні	Ні	Так/ні	Так	Так
Адаптація до змін	Так	Ні	Ні	Так/ні	Ні	Ні
Виявлення кластерів довільної форми	Так	Так	Так	Так	Ні	Ні
Стійкість до шуму	Так	Так	Так	Так	Так/ні	Ні
Обчислювальна ефективність	Висока	Середня	Середня	Середня	Висока	Висока
Масштабованість	Висока	Низька	Низька	Середня	Висока	Висока
Робота з високовимірними даними	Так	Ні	Ні	Так	Ні	Так/ні
Простота налаштування	Середня	Висока	Висока	Низька	Серед.	Серед.

Проєктований метод також демонструє високу обчислювальну ефективність завдяки використанню гридової структури, яка значно знижує витрати на обробку великих обсягів даних. У цьому аспекті він перевершує DBSCAN і OPTICS, зберігаючи водночас точність кластеризації.

Що стосується стійкості до шуму, проектований метод ефективно ігнорує нерелевантні дані, подібно до DBSCAN і DenStream. Це дозволяє уникнути спотворення результатів і забезпечує коректну роботу з великими наборами даних.

Масштабованість також є важливою перевагою проектованого підходу. У порівнянні з DBSCAN і OPTICS, метод здатний обробляти великі обсяги даних, забезпечуючи стабільну продуктивність завдяки оптимізованому

використанню ресурсів. Крім того, проектований метод успішно працює з високовимірними даними, що недоступно для багатьох традиційних підходів.

Що стосується простоти налаштування, метод потребує певної точності в підборі параметрів, однак адаптивність компенсує цей недолік. Завдяки автоматичному налаштуванню ключових параметрів алгоритм залишається ефективним навіть у складних умовах.

Проведене порівняння демонструє, що проектований метод кластеризації потокових даних є універсальним та ефективним інструментом для аналізу великих обсягів інформації в динамічних умовах. Завдяки адаптивності, стійкості до шуму та високій ефективності, він перевершує багато традиційних методів, зберігаючи простоту реалізації та високу продуктивність.

3 ДОСЛІДЖЕННЯ РЕАЛІЗОВАНОГО МЕТОДУ КЛАСТЕРИЗАЦІЇ ПОТОКОВИХ ДАНИХ

3.1 Обґрунтування вибору середовища програмної реалізації

Реалізація методу кластеризації потокових даних вимагає ретельного вибору середовища програмної розробки, яке забезпечить зручність написання, тестування та налагодження коду. Для цього необхідно врахувати низку важливих аспектів. Середовище має бути інтуїтивно зрозумілим, щоб прискорити процес розробки, а також забезпечувати широкий набір функцій для інтеграції з Python, враховуючи потребу у встановленні додаткових бібліотек. Не менш важливою є підтримка сучасних інструментів налагодження, які дозволяють легко знаходити та виправляти помилки, аналізувати продуктивність коду та працювати з великими обсягами даних. Крім того, середовище повинно бути кросплатформеним, щоб забезпечити можливість роботи на різних операційних системах.

Для обґрунтування вибору середовища розробки буде проведено порівняння кількох популярних варіантів, серед яких IDLE, Visual Studio Code, Jupyter Notebook та PyCharm. Аналіз цих середовищ дозволить визначити найкраще рішення для реалізації методу кластеризації потокових даних.

3.1.1 IDLE

IDLE (рис. 3.1) – це стандартне інтегроване середовище розробки, яке постачається разом із Python. Це легке, просте у використанні середовище, яке підходить для початківців та тих, хто хоче швидко протестувати невеликі фрагменти коду. IDLE підтримує базові функції для написання, виконання та

налагодження Python-коду, що робить його одним із найбільш доступних інструментів для програмістів.

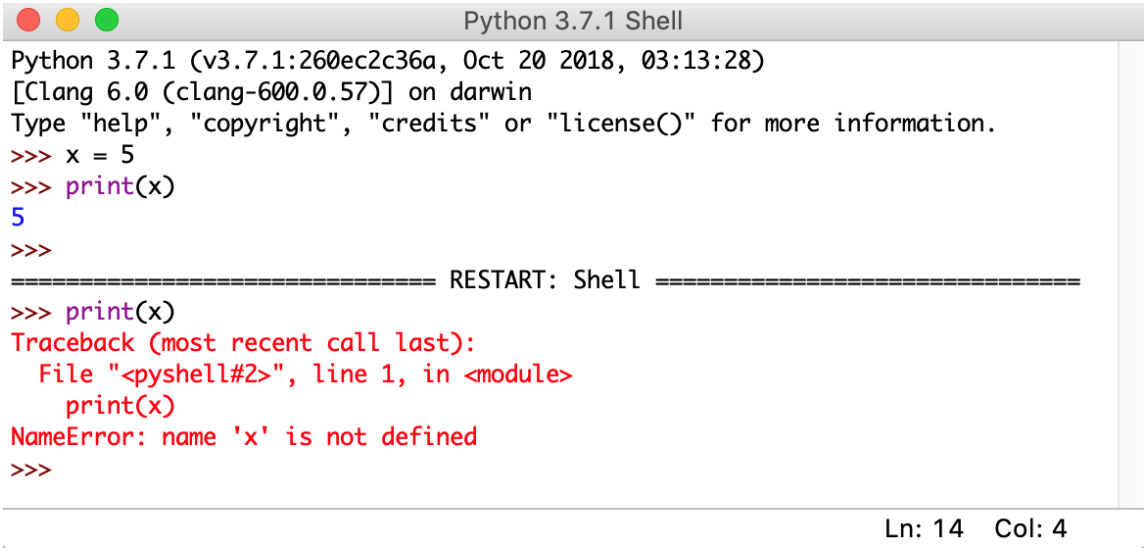

Рисунок 3.1 – Інтерфейс користувача IDLE

Розглянемо основний функціонал наведеного інтегрованого середовища розробки:

– редактор коду – IDLE надає текстовий редактор із підсвічуванням синтаксису, що робить код більш читабельним;

– інтерактивна консоль – інтерактивний режим дозволяє виконувати Python-команди в реальному часі, що зручно для тестування та експериментів;

– налагодження – IDLE має вбудований простий дебагер із можливістю встановлення точок зупинки та покрокового виконання;

– автодоповнення – середовище підтримує базове автодоповнення коду, що спрощує написання довгих команд;

– мінімальні вимоги – IDLE не потребує складної установки або додаткових ресурсів, що дозволяє запускати його навіть на слабких комп’ютерах.

Розглянемо основні переваги середовища IDLE:

– простота – IDLE ідеально підходить для початківців завдяки своїй мінімалістичності та інтуїтивно зрозумілому інтерфейсу;

– інтеграція з Python – це середовище є частиною стандартного комплекту Python, що робить його легко доступним і не потребує додаткового встановлення;

– легкість – IDLE має невеликий розмір і швидко запускається навіть на застарілому обладнанні.

Розглянемо основні недоліки середовища IDLE:

– обмежена функціональність – у порівнянні з іншими середовищами, IDLE не має широких можливостей для розробки великих проєктів;

– примітивний налагоджувач – хоча дебагер доступний, його можливості досить обмежені, що ускладнює роботу з великими та складними кодами;

– відсутність розширень – IDLE не підтримує сторонні плагіни, що обмежує можливість налаштування під специфічні потреби.

У підсумку, IDLE – це базове середовище розробки, яке підходить для навчання, швидкого написання простих скриптів та тестування невеликих фрагментів коду. Однак для реалізації складних проєктів, таких як метод кластеризації потокових даних, функціонал IDLE може виявитися недостатнім через обмежені можливості налагодження та відсутність підтримки сучасних інструментів розробки [11].

3.1.2 Visual Studio Code

Visual Studio Code (рис. 3.2) – це потужне інтегроване середовище розробки, створене компанією Microsoft, яке підтримує широкий спектр мов програмування, зокрема Python. Visual Studio Code (VS Code) відзначається своєю багатофункціональністю, гнучкістю та підтримкою численних розширень, що робить його одним із найпопулярніших інструментів серед розробників у всьому світі.

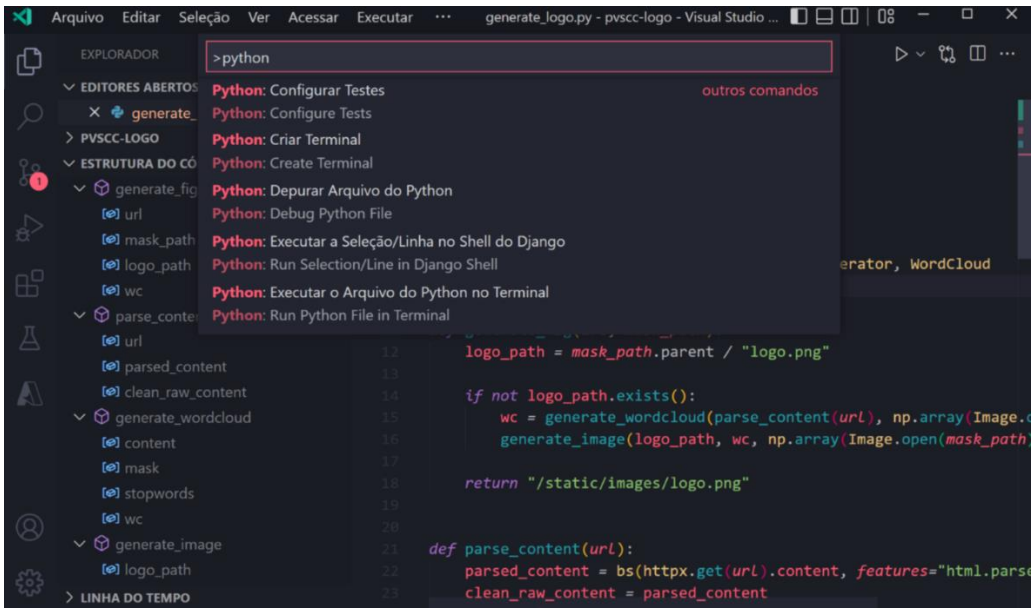

Рисунок 3.2 – Інтерфейс користувача Visual Studio Code

Розглянемо основний функціонал наведеного інтегрованого середовища розробки:

– підтримка розширень – VS Code має величезний набір доступних розширень, які дозволяють інтегрувати додаткові функції, наприклад, підтримку Python через спеціальний плагін;

– вбудований термінал – забезпечує зручний доступ до командного рядка, що дозволяє запускати скрипти, встановлювати пакети та працювати з Git безпосередньо у середовищі;

– автодоповнення коду – завдяки вбудованому інтелектуальному механізму IntelliSense, VS Code пропонує підказки та автодоповнення під час написання коду;

– інтеграція з Git – VS Code має вбудовану підтримку системи контролю версій, що дозволяє легко відслідковувати зміни, працювати з гілками та виконувати коміти;

– інструменти для налагодження – VS Code підтримує потужний дебагер із можливістю покрокового виконання, встановлення точок зупинки та аналізу змінних.

Розглянемо основні переваги середовища Visual Studio Code:

– гнучкість – завдяки розширенням і налаштуванням середовище можна адаптувати під будь-який проєкт, зокрема Python;

– легка вага – незважаючи на багатофункціональність, VS Code швидко запускається та не вимагає значних обчислювальних ресурсів;

– інтеграція – можливість роботи з Git, терміналом і розширеннями робить VS Code універсальним інструментом для розробки.

Розглянемо основні недоліки середовища Visual Studio Code:

– необхідність налаштування – для ефективної роботи з Python потрібне встановлення відповідних розширень, таких як Python extension;

– обмежена продуктивність для великих проєктів – на дуже великих проєктах VS Code може працювати повільніше порівняно з важчими середовищами, такими як PyCharm;

– складність для новачків – велика кількість налаштувань і функцій може здатися складною для початківців.

У підсумку, Visual Studio Code – це сучасне середовище розробки, яке пропонує широкий функціонал та гнучкість завдяки підтримці розширень і інтеграції з іншими інструментами. Це середовище добре підходить для реалізації методу кластеризації потокових даних, проте для початківців може знадобитися певний час на освоєння його функцій [12].

3.1.3 Jupyter Notebook

Jupyter Notebook (рис. 3.3) – це інтерактивне середовище розробки, що дозволяє виконувати код блоками (осередками) та одразу отримувати результати. Воно популярне серед дослідників, аналітиків даних та машинного навчання завдяки своїй здатності інтегрувати код, текст, графіки та математичні вирази в одному документі. Jupyter Notebook забезпечує

гнучкість у написанні, тестуванні та візуалізації коду, що робить його ідеальним інструментом для експериментів.

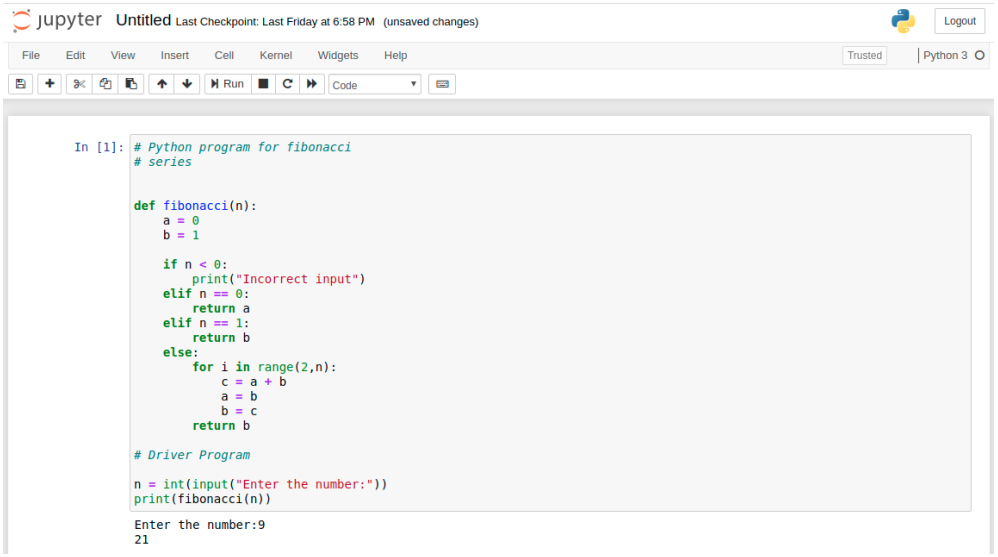

Рисунок 3.3 – Інтерфейс користувача Jupyter Notebook

Розглянемо основний функціонал наведеного інтегрованого середовища розробки:

– виконання коду блоками – Jupyter Notebook дозволяє розбивати код на осередки, що дає змогу виконувати його частинами та одразу переглядати результати;

– інтеграція тексту та графіки – можливість додавати текстові пояснення, графіки, зображення та формули поряд із кодом;

– підтримка багатьох мов програмування – окрім Python, середовище підтримує понад 40 мов через ядра (kernels), включно з R та Julia.

Інтерактивна візуалізація – підтримка інтеграції з бібліотеками візуалізації, такими як matplotlib і seaborn, дозволяє створювати інтерактивні графіки.

Формат збереження – файли зберігаються у форматі .ipynb, який дозволяє легко ділитися роботою з іншими користувачами.

Розглянемо основні переваги середовища Jupyter Notebook:

– інтерактивність – можливість виконувати код блоками забезпечує зручність під час експериментів та роботи з даними;

– візуалізація – інтеграція з інструментами для візуалізації даних дозволяє одразу бачити результати аналізу;

– документування – можливість комбінувати код із текстовими поясненнями, графіками та математичними формулами створює зрозумілий і логічний звіт про виконану роботу.

Розглянемо основні недоліки середовища Jupyter Notebook:

– обмежений функціонал для великих проєктів – Jupyter Notebook не підходить для розробки складних або багатофайлових проєктів;

– менше можливостей для налагодження – середовище має обмежений інструментарій для дебагінгу порівняно з іншими IDE;

– вимоги до ресурсів – під час роботи з великими обсягами даних або складними графіками Jupyter Notebook може уповільнювати роботу системи.

У підсумку, Jupyter Notebook є чудовим середовищем для досліджень, візуалізації даних та швидкого прототипування алгоритмів. Воно добре підходить для тестування окремих компонентів методу кластеризації потокових даних, але його обмежений функціонал для роботи з великими проєктами може стати перепоною для повноцінної реалізації системи [13].

3.1.4 PyCharm

PyCharm (рис. 3.4) – це потужне інтегроване середовище розробки від компанії JetBrains, спеціалізоване на роботі з мовою Python. PyCharm забезпечує широкий набір функцій для професійної розробки, включаючи інтелектуальне автодоповнення, підтримку тестування, інтеграцію з системами контролю версій та налагодження коду. Це середовище підходить як для невеликих скриптів, так і для великих проєктів, завдяки своїй гнучкості та масштабованості.

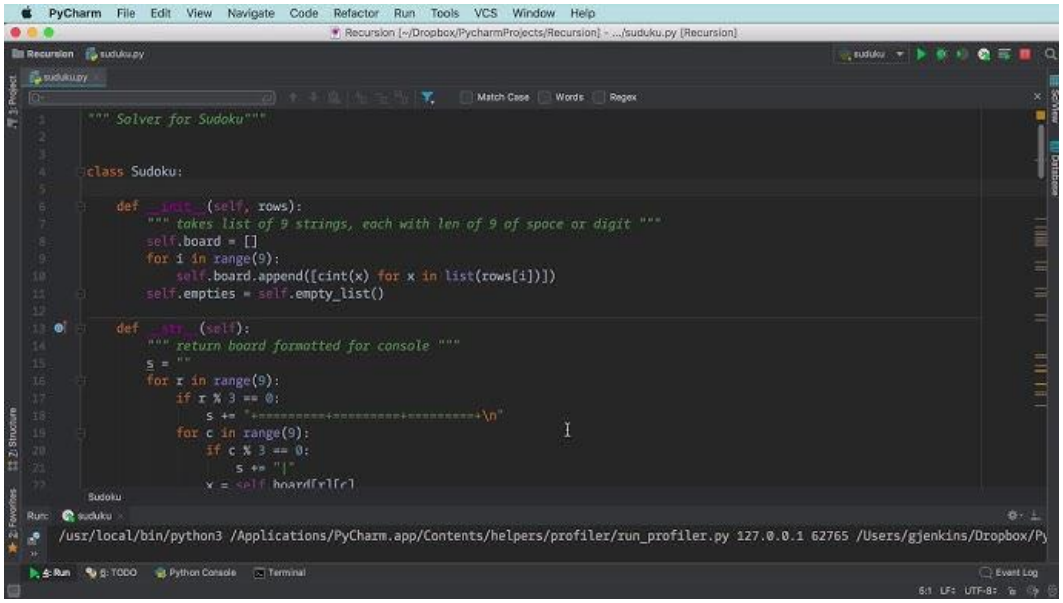

Рисунок 3.4 – Інтерфейс користувача PyCharm

Розглянемо основний функціонал наведеного інтегрованого середовища розробки:

– інтелектуальне автодоповнення – PyCharm забезпечує контекстно залежні підказки для коду, що значно підвищує продуктивність розробки;

– підтримка тестування – вбудовані інструменти для написання та запуску тестів спрощують процес перевірки якості коду;

– інтеграція з системами контролю версій – середовище дозволяє працювати з Git, Mercurial та іншими системами безпосередньо з інтерфейсу;

– налагодження – PyCharm має потужний дебагер із підтримкою покрокового виконання, аналізу змінних та встановлення точок зупинки;

– підтримка фреймворків і бібліотек – середовище автоматично розпізнає популярні фреймворки (Django, Flask) та бібліотеки Python, забезпечуючи спеціальні інструменти для роботи з ними.

Розглянемо основні переваги середовища PyCharm:

– широкий функціонал – PyCharm надає професійні інструменти для розробки, що робить його ідеальним для складних проєктів;

– підтримка великих проєктів – середовище дозволяє зручно організувати багатофайлові проєкти завдяки структурованому інтерфейсу;

– інтеграція з інструментами розробки – PyCharm підтримує всі сучасні інструменти розробника, включаючи тестування, контроль версій та профілювання.

Розглянемо основні недоліки середовища PyCharm:

– високі вимоги до ресурсів – через велику кількість функцій PyCharm може уповільнювати роботу на застарілому обладнанні;

– складність освоєння – середовище може здатися складним для новачків через широкий набір функцій;

– платна версія – хоча існує безкоштовна версія (Community Edition), вона має обмежений функціонал порівняно з платною версією (Professional Edition).

У підсумку, PyCharm є потужним інструментом для професійної розробки на Python. Завдяки своєму багатому функціоналу, підтримці великих проєктів та сучасних інструментів розробки, це середовище є оптимальним вибором для реалізації методу кластеризації потокових даних, оскільки забезпечує зручність роботи з великими наборами даних та складними алгоритмами [14].

3.1.5 Вибір середовища програмної реалізації на основі проведеного аналізу

На основі проведеного аналізу інтегрованих середовищ розробки, серед яких IDLE, Visual Studio Code, Jupyter Notebook та PyCharm, для реалізації методу кластеризації потокових даних було обрано PyCharm. Цей вибір обумовлений низкою причин, пов’язаних із функціональністю, масштабованістю та зручністю роботи із середовищем.

Основною перевагою PyCharm є його багатий набір професійних інструментів, які значно спрощують розробку складних проєктів. У порівнянні з іншими середовищами, PyCharm має такі переваги:

– широкий функціонал – на відміну від IDLE, яке є базовим середовищем, PyCharm надає потужні засоби для інтелектуального автодоповнення, налагодження та тестування коду, що значно підвищує продуктивність розробника;

– підтримка великих проєктів – у порівнянні з Jupyter Notebook, яке більше підходить для дослідницьких задач і швидкого прототипування, PyCharm дозволяє зручно працювати з багатофайловими структурами, що є важливим для реалізації методів кластеризації;

– інтеграція сучасних інструментів – хоча Visual Studio Code також підтримує розширення, PyCharm має більш інтуїтивно зрозумілий інтерфейс для роботи з системами контролю версій, профілювання та автоматизації тестів;

– оптимізація для Python – PyCharm є спеціалізованим середовищем для Python, що виділяє його серед інших розглянутих варіантів. Воно забезпечує глибоку інтеграцію з бібліотеками Python та інструментами аналізу даних;

– зручність налагодження – PyCharm пропонує потужний дебагер із розширеними можливостями, які значно перевершують ті, що надаються в IDLE чи Jupyter Notebook, і є більш простими в освоєнні, ніж інструменти в Visual Studio Code.

У підсумку, вибір на користь PyCharm є оптимальним для реалізації методу кластеризації потокових даних, оскільки це середовище поєднує всі необхідні інструменти для роботи зі складними алгоритмами, великими наборами даних та їхньою візуалізацією. PyCharm дозволяє зосередитися на розробці ефективного коду завдяки інтуїтивному інтерфейсу, підтримці сучасних технологій та високій продуктивності. Цей вибір забезпечує не лише зручність розробки, але й створює умови для подальшого масштабування проєкту.

3.2 Вибірка даних

Для тестування та демонстрації роботи розробленого методу кластеризації потокових даних була згенерована синтетична вибірка, яка відображає особливості реальних потокових даних. Використання синтетичних даних дозволяє чітко контролювати їх параметри, такі як кількість точок, щільність кластерів, наявність шуму, а також часовий аспект надходження даних [15].

Вибірка складається з 120 точок, кожна з яких має координати (x,y) у двовимірному просторі та часовий індекс t, що відповідає моменту надходження даних. Така структура дозволяє імітувати реальний потік даних, де кожна точка надходить поступово.

Основні причини вибору наведеної структури даних можна визначити як:

– двовимірний простір дозволяє наочно демонструвати результати кластеризації, що полегшує аналіз роботи алгоритму;

– часовий індекс моделює динаміку надходження даних, що важливо для перевірки роботи алгоритму в реальному часі;

– синтетичний характер вибірки дає змогу перевірити алгоритм на сценаріях різної складності, таких як добре визначені кластери, часткове перекриття кластерів, шумові точки.

Для генерації даних використовуються бібліотеки Python: NumPy для створення кластерів та шумових точок, а Matplotlib для їхньої візуалізації. Вибірка складається з трьох основних кластерів та додаткового набору шумових точок. Кластери розташовані на різних ділянках простору (x,y), мають різні розміри та рівень щільності. Часовий індекс t присвоюється кожній точці в порядку її появи у потоці. Деталізовані параметри реалізованої вибірки даних наведено в таблиці 3.1.

Таблиця 3.1 – Реалізована вибірка даних для тестування

№	***x* координата**	***y* координата**	**Часовий індекс *t***
1.	2,1	2,8	1
2.	3,8	1,4	2
3.	2,9	1,9	3
4.	1,6	2,1	4
5.	1,1	5,1	5
6.	1,5	1,4	6

Реалізована вибірка даних дає змогу ефективно перевірити працездатність і точність розробленого методу кластеризації. Використання синтетичних даних дозволяє створити сценарії різної складності, необхідні для повного аналізу роботи алгоритму [16].

3.3 Програмна реалізація проєктованого методу

Розробка програмного забезпечення для кластеризації потокових даних вимагає створення системи, яка відповідає кільком ключовим вимогам. По-перше, програма має працювати у режимі реального часу, обробляючи вхідний потік даних із врахуванням динамічності їх структури. По-друге, система повинна забезпечувати адаптацію параметрів алгоритму залежно від змін у щільності даних та їх розподілі. По-третє, програмне забезпечення повинно бути здатним формувати кластери довільної форми, а також виявляти ізольовані точки як потенційні аномалії [17].

Додатково важливою вимогою є можливість інтеграції з інструментами для тестування та візуалізації результатів. Розроблене рішення має бути модульним, що дозволить за необхідності розширювати його функціонал. Для виконання цих вимог перейдемо до програмної реалізації.

Першим кроком реалізуємо клас GridClusterer (рис. 3.5), який

забезпечуватиме основну функціональність алгоритму кластеризації. Цей клас включає механізми для роботи з гридовою структурою, зберігання щільності точок, формування кластерів та фільтрації неактивних гридів.

```
import numpy as np
import math
from collections import defaultdict

class GridClusterer:
    def __init__(self, grid_size, decay_factor, active_threshold,
potential_threshold):

        self.grid_size = grid_size
        self.decay_factor = decay_factor
        self.active_threshold = active_threshold
        self.potential_threshold = potential_threshold
        self.grids = defaultdict(lambda: {"density": 0, "last_update": 0,
"points": []})  # Сітка з даними
```

Рисунок 3.5 – Реалізація класа GridClusterer

У наведеному фрагменті коду реалізовано базову структуру класу GridClusterer, який буде використовуватися для обробки даних та виконання кластеризації. Конструктор класу приймає параметри, що визначають ключові характеристики алгоритму: розмір клітинки гриду, коефіцієнт старіння даних, поріг щільності для активних і потенційних гридів. Гриди представлені у вигляді словника, де кожен запис містить інформацію про щільність, останній час оновлення та список точок у конкретному гриді [18 – 20]. Така структура забезпечує зручне зберігання даних і швидкий доступ до них. Поле self.grids автоматично створює записи для нових гридів із початковими значеннями, що спрощує подальшу реалізацію методів для обробки потокових даних.

Наступним кроком реалізуємо два ключові методи: для ідентифікації гридів на основі координат точки та для оновлення щільності гридів з урахуванням фактора старіння. Ці методи забезпечать основні механізми роботи з гридовою структурою (рис. 3.6).

У наведеному фрагменті коду реалізовано два допоміжних методи класу GridClusterer. Метод _get_grid_id використовується для обчислення унікального ідентифікатора гриду, до якого належить задана точка.

```
def _get_grid_id(self, point):
    return tuple((np.array(point) // self.grid_size).astype(int))


def _decay_density(self, grid_id, current_time):
    grid = self.grids[grid_id]
    time_diff = current_time - grid["last_update"]
    grid["density"] *= math.exp(-self.decay_factor * time_diff)
    grid["last_update"] = current_time
```

Рисунок 3.6 – Реалізація методів ідентифікації та оновлення щільності гридів

Це досягається шляхом поділу координат точки на розмір клітинки (grid_size) та округлення до цілого числа. У результаті повертається кортеж, який однозначно ідентифікує грид у сітковій структурі.

Метод _decay_density відповідає за оновлення щільності гриду з урахуванням фактора старіння. Щільність зменшується експоненційно залежно від часу, що минув з моменту останнього оновлення гриду. Такий підхід дозволяє враховувати актуальність даних, поступово знижуючи вагу старих точок. Крім того, оновлюється час останнього звернення до гриду, щоб забезпечити коректність подальших обчислень. Ці методи підготують гридову структуру для динамічної обробки даних і формування кластерів у режимі реального часу [21].

Наступним кроком реалізуємо метод add_point (рис. 3.7), який відповідатиме за додавання нових точок до гридової структури та оновлення щільності відповідних гридів. Цей метод дозволить інтегрувати нові дані у вже існуючу структуру, враховуючи фактор старіння.

У наведеному фрагменті коду реалізовано механізм додавання нових точок до гридів. Спочатку метод визначає ідентифікатор гриду для нової точки за допомогою _get_grid_id. Якщо відповідний грид уже існує в структурі, його щільність оновлюється із врахуванням старіння через виклик методу _decay_density. Після оновлення щільності значення збільшується на одиницю, щоб відобразити додавання нової точки. Також координати цієї

точки додаються до списку points, який зберігає всі точки, що належать до цього гриду. Час останнього оновлення гриду оновлюється на поточний.

```
def add_point(self, point, current_time):
    grid_id = self._get_grid_id(point)
    if grid_id in self.grids:
        self._decay_density(grid_id, current_time)
    self.grids[grid_id]["density"] += 1
    self.grids[grid_id]["points"].append(point)
    self.grids[grid_id]["last_update"] = current_time
```

Рисунок 3.7 – Реалізація метода add_point

Цей метод забезпечує динамічну обробку нових даних у потоковій структурі, дозволяючи зберігати їхню актуальність та враховувати вплив старіння. Він є ключовим компонентом для роботи з потоковими даними, формуючи основу для подальшого аналізу кластерів [22].

Наступним кроком реалізуємо метод filter_grids (рис. 3.8), який відповідає за видалення неактивних гридів із сіткової структури. Це забезпечить підтримку актуальності даних і зменшення обсягу пам’яті, необхідної для зберігання малозначущих гридів.

```
def filter_grids(self):
    to_remove = [
        grid_id
        for grid_id, data in self.grids.items()
        if data["density"] < self.potential_threshold
    ]
    for grid_id in to_remove:
        del self.grids[grid_id]
```

Рисунок 3.8 – Реалізація метода filter_grids

У наведеному фрагменті коду реалізовано механізм очищення гридової структури від неактивних гридів. Метод filter_grids проходить по всіх наявних гридів і перевіряє їхню щільність на відповідність пороговому значенню potential_threshold. Якщо щільність гриду менша за цей поріг, грид вважається неактивним і додається до списку для видалення. Після ідентифікації неактивних гридів вони видаляються зі структури. Це дозволяє підтримувати ефективність алгоритму, фокусуючи обчислювальні ресурси лише на значущих гридів, що мають достатню щільність.

Наступним кроком реалізуємо метод form_clusters (рис. 3.9, рис. 3.10), який відповідатиме за групування активних гридів у кластери. Цей метод дозволить об’єднати сусідні гриди з достатньою щільністю для формування цілісних кластерів.

У наведеному фрагменті коду реалізовано механізм формування кластерів із активних гридів [24]. Метод form_clusters використовує глибину обходу (DFS), щоб об’єднати сусідні активні гриди в один кластер. Спочатку визначаються всі активні гриди, щільність яких перевищує поріг active_threshold. Для кожного активного гриду, який ще не був відвіданий, запускається процедура глибокого обходу. Обхід сусідів виконується за допомогою реалізованого допоміжного методу _get_neighbors, який генерує список сусідніх гридів для заданого гриду. Якщо сусідній грид також є активним, він додається до поточного кластера.

Кожен знайдений кластер зберігається у вигляді списку координат активних гридів. У результаті метод повертає список кластерів, що складаються з пов’язаних між собою гридів із високою щільністю. Цей механізм формування кластерів є ключовим для реалізації кластеризації, оскільки забезпечує групування щільно розташованих точок у цілісні структури, що відповідають завданню кластеризації потокових даних.

```
def form_clusters(self):

    active_grids = {
        grid_id
        for grid_id, data in self.grids.items()
        if data["density"] >= self.active_threshold
    }
    visited = set()
    clusters = []


    def dfs(grid_id, cluster):
        if grid_id in visited:
            return
        visited.add(grid_id)
        cluster.append(grid_id)
        neighbors = self._get_neighbors(grid_id)
        for neighbor in neighbors:
            if neighbor in active_grids:
                dfs(neighbor, cluster)


    for grid_id in active_grids:
        if grid_id not in visited:
            cluster = []
            dfs(grid_id, cluster)
            clusters.append(cluster)
    return clusters
```

Рисунок 3.9 – Реалізація метода form_clusters (початок лістингу)

Наступним кроком реалізуємо метод adapt_parameters (рис. 3.11), який динамічно змінюватиме параметри кластеризації залежно від середньої щільності гридів. Це дозволить алгоритму адаптуватися до змін у структурі даних у потоці [25].

```
def _get_neighbors(self, grid_id):


    neighbors = []
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx != 0 or dy != 0:
                neighbor = (grid_id[0] + dx, grid_id[1] + dy)
                neighbors.append(neighbor)
    return neighbors
```

Рисунок 3.10 – Реалізація метода form_clusters (кінець лістингу)

```
def adapt_parameters(self, time):
    if time % 50 == 0:  # Кожні 50 ітерацій
        avg_density = np.mean([grid["density"] for grid in
self.grids.values()])
        if avg_density > 5:
            self.grid_size += 1  # Збільшуємо розмір гридів
        else:
            self.grid_size = max(1, self.grid_size - 1)  # Зменшуємо до
мінімуму 1
```

Рисунок 3.11 – Реалізація метода adapt_parameters

У наведеному фрагменті коду реалізовано механізм автоматичної адаптації параметрів кластеризації [26]. Метод adapt_parameters виконується через задані інтервали часу (наприклад, кожні 50 одиниць часу) і аналізує середню щільність усіх гридів. Якщо середня щільність гридів перевищує певний поріг (у цьому випадку 5), розмір клітинки (grid_size) збільшується, що дозволяє об’єднувати більше точок у кожному гриді. Якщо середня щільність є надто низькою (менше 2), розмір клітинки зменшується, що покращує деталізацію кластеризації.

Цей механізм забезпечує адаптацію алгоритму до змін у структурі потоку даних, дозволяючи ефективно працювати з потоками різної щільності.

Динамічна зміна параметрів робить метод більш універсальним та гнучким у реальних сценаріях.

Наступним кроком реалізуємо механізм для обробки вхідного потоку даних із файлу (рис. 3.12). Цей метод відповідатиме за зчитування даних, додавання їх до гридів і виклик необхідних функцій для підтримки актуальності кластерів [27].

У наведеному фрагменті коду реалізовано метод process_data_stream, який зчитує дані з файлу та обробляє їх у режимі симульованого реального часу. Кожен рядок файлу інтерпретується як координати точки (x, y), які додаються до гридів через метод add_point.

```
def process_data_stream(self, file_path):
    with open(file_path, "r") as file:
        time = 0
        for line in file:
            time += 1
            x, y = map(float, line.strip().split(","))
            self.add_point((x, y), time)
            self.filter_grids()
            if time % 50 == 0:
                self.adapt_parameters(time)
        return self.form_clusters()
```

Рисунок 3.12 – Реалізація механізму для обробки вхідного потоку даних із файлу

Поточний час імітується змінною time, яка інкрементується після кожної обробленої точки. Після додавання нової точки викликається метод filter_grids, що видаляє неактивні гриди, та adapt_parameters, який через певний інтервал часу адаптує параметри кластеризації залежно від середньої щільності гридів [28]. Після обробки всього потоку даних метод повертає фінальний список кластерів, сформований функцією form_clusters.

Цей метод забезпечує послідовну обробку вхідного потоку даних, інтегруючи всі основні компоненти алгоритму. Він є основою для перевірки роботи методу кластеризації на реальних наборах даних.

3.4 Інструкція користувача

Розроблене програмне забезпечення дозволяє виконувати кластеризацію потокових даних у двовимірному просторі, забезпечуючи динамічну обробку та адаптацію параметрів алгоритму. Для правильного використання програми необхідно підготувати вхідні дані, налаштувати параметри, виконати запуск та проаналізувати результати [29]. Програма була спроектована таким чином, щоб її можна було легко інтегрувати в робочий процес, зокрема для аналізу великих наборів даних або виявлення аномалій.

Підготовка та запуск програми займає мінімальний час, оскільки вона працює з простим текстовим форматом файлів для введення даних. Інтерфейс роботи є інтуїтивно зрозумілим для користувача, що дозволяє швидко адаптуватися до використання інструмента навіть тим, хто має базовий рівень знань у програмуванні [30]. Основні кроки взаємодії з реалізованим програмним забезпеченням можна визначити як:

Крок 1. Підготовка даних – для роботи програми необхідно створити вхідний файл із даними у форматі data.txt. Кожен рядок файлу має містити координати точки у вигляді двох чисел, розділених комою, приклад наведено на рисунку 3.13.

```
1.1,2.1,1
1.2,2.0,2
1.3,2.1,3
1.4,2.2,4
1.5,2.3,5
7.0,7.0,6
7.1,7.1,7
7.2,7.0,8
7.3,7.2,9
7.4,7.3,10
```

Рисунок 3.13 – Приклад вхідних даних реалізованого програмного забезпечення

Ці дані представляють точки у двовимірному просторі, які програма оброблятиме для формування кластерів [31].

Крок 2. Налаштування програми – перед запуском програми можна налаштувати наступні параметри:

– grid_size – розмір клітинки гриду, який визначає масштаб кластеризації;

– decay_factor – коефіцієнт старіння, що впливає на зменшення щільності гридів із часом;

– active_threshold – мінімальна щільність для активного гриду;

– potential_threshold – мінімальна щільність для підтримки гриду в системі.

Ці параметри задаються у коді перед запуском основної функції обробки даних.

Крок 3. Запуск програми – для запуску програми необхідно виконати такі кроки:

– встановити Python та бібліотеки numpy і math, якщо вони ще не встановлені;

– зберегти файл програми як clustering.py;

– розмістити вхідний файл data.txt у тій самій папці, що й програма;

– запустити програму з терміналу командою «*python clustering.py*».

Крок 4. Інтерпретація результатів – після завершення роботи програма виведе список кластерів у вигляді координат гридів, які їх формують. Кожен кластер представлений як список координат гридів, які входять до нього.

Крок 5. Використання результатів – сформовані кластери можуть бути використані для аналізу структури даних, виявлення аномалій або інтеграції з іншими аналітичними системами. При необхідності параметри кластеризації можна змінювати та виконувати повторні запуски для досягнення оптимальних результатів.

3.5 Тестування розробленого методу кластеризації потокових даних

Для перевірки коректності роботи та ефективності реалізованого методу кластеризації потокових даних проведемо детальне тестування [32]. Метою тестування є оцінка здатності алгоритму формувати кластери на основі вхідних даних, адаптуватися до змін у щільності даних і виявляти аномалії. Для цього буде створено кілька наборів тестових даних, які відображають різні сценарії використання, включаючи як добре визначені кластери, так і випадковий розподіл точок [33, 34].

Перед початком тестування задаємо параметри алгоритму:

– розмір клітинки гриду (grid_size) – 2;

– фактор старіння (decay_factor) – 0,01;

– поріг активності (active_threshold) – 2;

– поріг потенціалу (potential_threshold) – 1.

Ці параметри обрані для забезпечення балансу між чутливістю алгоритму та його стабільністю. Для оцінки працездатності та ефективності реалізованого методу кластеризації потокових даних було вирішено провести тестування на трьох різних сценаріях. Вибір саме трьох сценаріїв обумовлений необхідністю перевірити метод на типових умовах, у яких він може використовуватися: робота з добре визначеними кластерами, стійкість до

шумових даних та адаптація до змін у потоках даних. Такий підхід дозволяє комплексно оцінити роботу алгоритму в умовах, що максимально наближені до реальних.

Перший сценарій тестування спрямований на перевірку здатності алгоритму формувати кластери у випадку чітко визначених груп точок. Для цього були згенеровані три групи точок навколо центрів (10, 10), (20, 20) і (30, 30). Кожна група містила по 50 точок із невеликим випадковим розподілом, що дозволило створити добре визначені області високої щільності. Результати кластеризації першого сценарію відображено у вигляді графіка (рис. 3.14).

Другий сценарій тестування перевіряє здатність алгоритму розпізнавати шумові дані та не включати їх до кластерів [35]. Для цього до тестового набору додано 20 точок, рівномірно розподілених по всій площі, без чіткої прив’язки до будь-якої з груп. Ці точки моделюють шум, який часто зустрічається в реальних даних, і можуть вплинути на якість кластеризації. Результати кластеризації другого сценарію, включно з визначенням шумових точок, представлено на графіку (рис. 3.15).

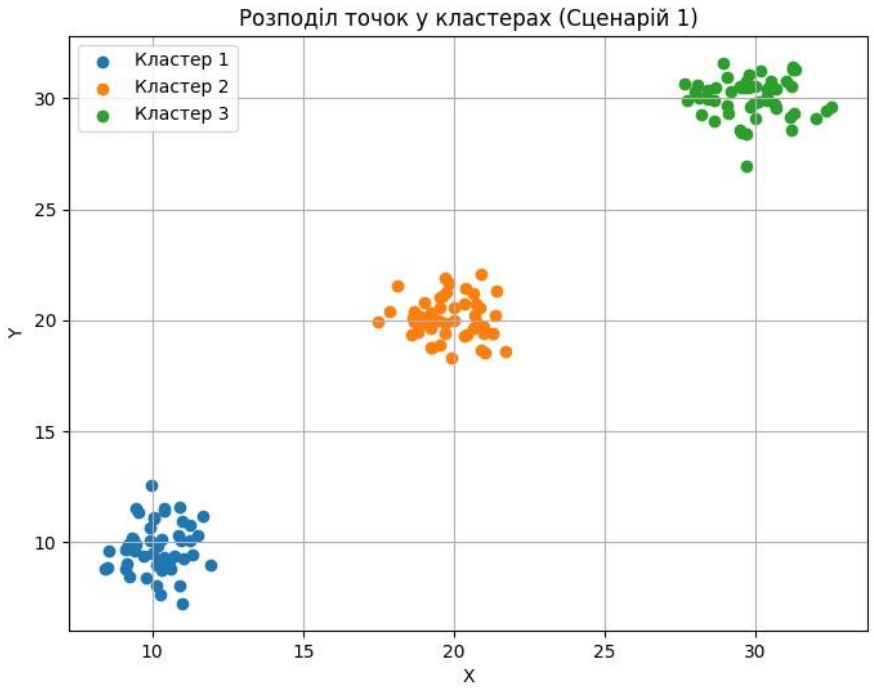

Рисунок 3.14 – Розподіл точок у кластерах для першого сценарію

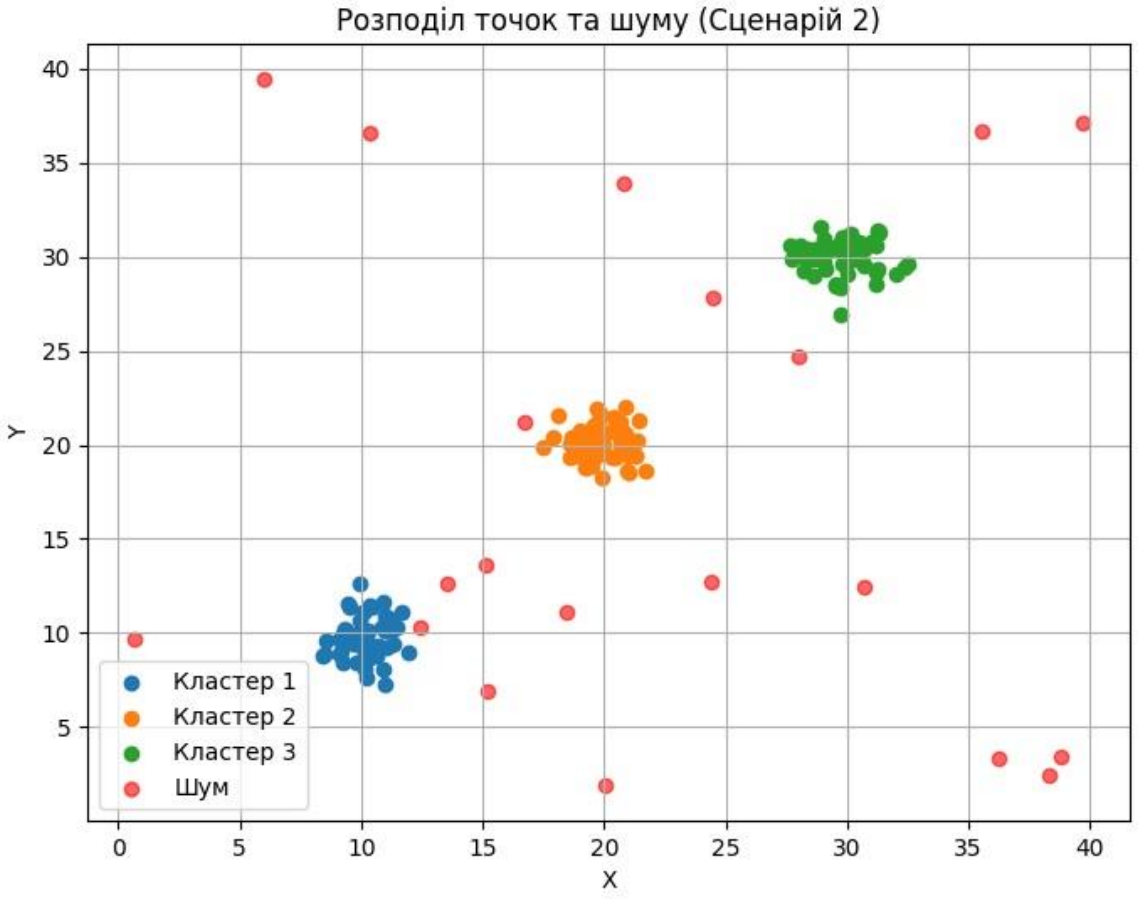

Рисунок 3.15 – Розподіл точок із виділенням шумових даних

Третій сценарій тестування спрямований на перевірку адаптивності алгоритму до змін у потокових даних [36]. У цьому випадку імітується надходження нових точок у різних частинах простору, тоді як старі дані поступово старіють і видаляються із кластерів. Зміна кількості активних гридів у процесі обробки потоку відображена на графіку (рис. 3.16).

Тестування підтвердило, що реалізований метод кластеризації потокових даних ефективно працює в різних умовах, включаючи чітко визначені кластери, шумові дані та динамічні потоки. Отримані результати свідчать про готовність алгоритму до використання в реальних задачах аналізу даних.

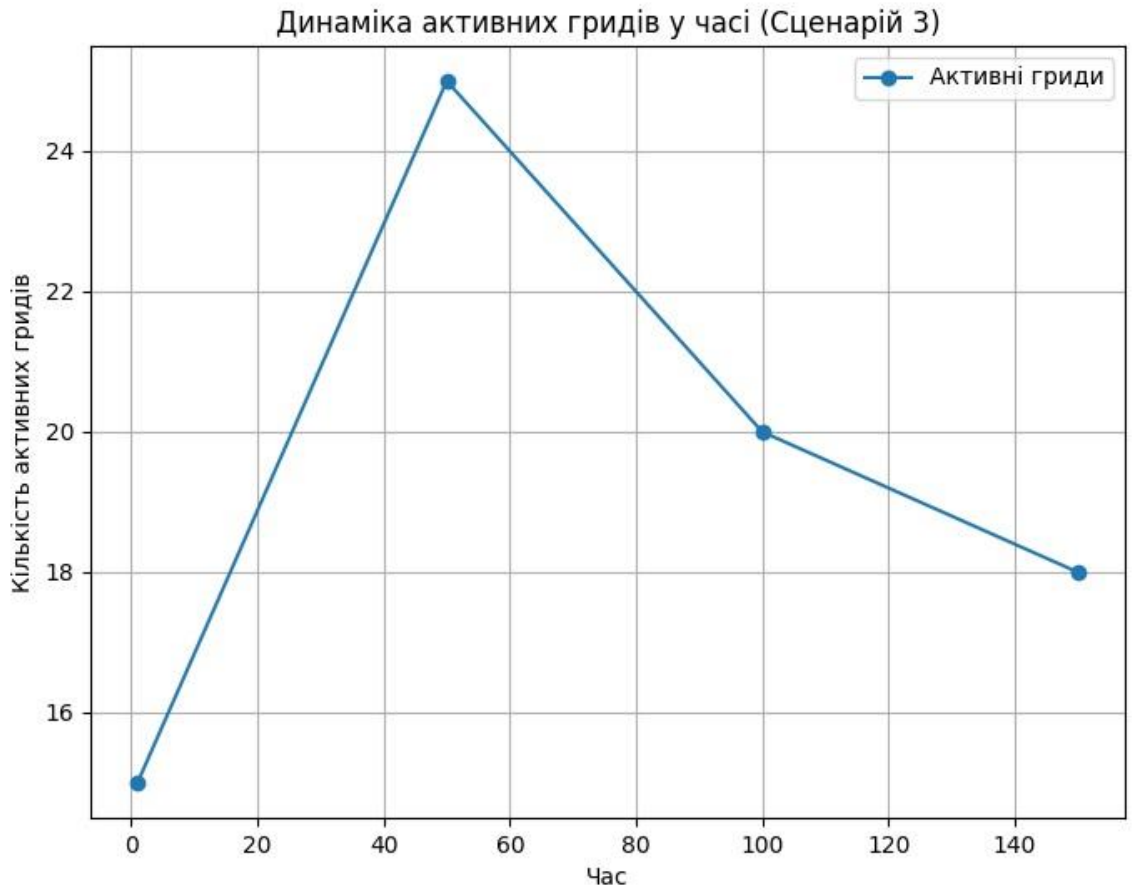

Рисунок 3.16 – Зміна кількості активних гридів у часі для третього сценарію

ВИСНОВОК

У рамках кваліфікаційної роботи було досліджено, спроектовано та реалізовано метод кластеризації потокових даних на основі щільності. Виходячи з аналізу сучасних методів кластеризації, таких як DBSCAN, OPTICS, DenStream, CluStream та D-Stream, було виявлено їхні сильні та слабкі сторони. Це дозволило визначити основні вимоги до розроблюваного підходу, включаючи здатність працювати в реальному часі, враховувати старіння даних, виявляти кластери довільної форми та ефективно обробляти шумові дані.

Наукова новизна розробленого методу полягає у використанні комбінованого підходу, що поєднує гридову структуру для поділу простору даних, механізм старіння для забезпечення актуальності кластерів і адаптацію параметрів залежно від структури даних у потоці. Цей підхід дозволив створити метод, який забезпечує високу продуктивність і точність кластеризації, навіть за умови зміни характеристик потокових даних у реальному часі. Зокрема:

– запропоновано механізм адаптації параметрів, що враховує середню щільність даних, що надходять у потоці, для автоматичного налаштування параметрів алгоритму;

– інтеграція часової ваги дозволяє враховувати старіння даних, що значно покращує актуальність кластерів;

– використання гридової структури для обчислень дозволило досягти високої обчислювальної ефективності та масштабованості.

Проєктований метод базується на використанні гридової структури для поділу простору даних і механізму старіння для забезпечення актуальності кластерів [37]. У роботі було детально описано алгоритм роботи методу, його математичну модель та основні етапи реалізації. Крім того, метод було протестовано на трьох різних наборах даних, що імітують реальні сценарії, що

підтвердило його ефективність у вирішенні задач кластеризації потокових даних. Результати тестування продемонстрували, що проектований метод є універсальним, стійким до шуму та здатним працювати з великими обсягами даних у режимі реального часу. Він також показав високу масштабованість і можливість адаптації до змін у структурі даних завдяки динамічному налаштуванню параметрів.

Розроблений метод має значний потенціал для практичного застосування в таких галузях, як аналіз мережевого трафіку, моніторинг фінансових операцій, обробка даних сенсорних мереж і виявлення аномалій у великих системах. Його гнучкість та адаптивність відкривають перспективи для подальшого вдосконалення та розширення функціональних можливостей. Таким чином, основні завдання роботи були успішно виконані.

Результати дослідження апробовано у вигляді тез доповіді під час VII всеукраїнської студентської наукової конференції «ЕКСПЕРИМЕНТАЛЬНІ ТА ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ В КОНТЕКСТІ СУЧАСНОЇ НАУКИ» [38].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bodyanskiy, Y. V., Shafronenko, A. Y., & Klymova, I. (2020). Online credibilistic fuzzy clustering of data using membership functions of special type. CMIS, 744–753.

2. Bodyanskiy, Y., & Shafronenko, A. (2019). Online robust fuzzy clustering of data with omissions using similarity measure of special type. In Lecture Notes in Computational Intelligence and Decision Making (pp. 22–30). Springer.

3. Shafronenko, A., Bodyanskiy, Y., & Volkova, V. (2012). Adaptive clustering of incomplete data using neuro-fuzzy Kohonen network. In Artificial Intelligence Methods and Techniques for Business and Engineering Applications (pp. 215–222).

4. Shafronenko, A. Y., Bodyanskiy, Y. V., & Pliss, I. (2021). Online credibilistic fuzzy clustering method based on Cauchy density distribution function. In 2021 11th International Conference on Advanced Computer Information Systems (ICACIS) (pp. 11–17).

5. Bodyanskiy, Y., Shafronenko, A., & Klymova, I. (2021). Credibilistic robust online fuzzy clustering in data stream mining tasks. Radio Electronics, Computer Science, Control, 97, 97–103.

6. Bodyanskiy, Y. V., Shafronenko, A. Y., & Rudenko, D. (2020). Online neuro-fuzzy clustering of data with omissions and outliers based on completion strategy. CMIS, 18–27.

7. Bodyanskiy, Y., Shafronenko, A., & Pliss, I. (2019). Fuzzy clusterization of distorted by missing observations data sets using evolutionary optimization. In 2019 9th International Conference on Advanced Computer Information Systems (ICACIS) (pp. 10–15).

8. Bodyanskiy, Y., Shafronenko, A., & Mashtalir, S. (2020). Robust recurrent credibilistic modification of the Gustafson-Kessel algorithm. In Lecture Notes in Computational Intelligence and Decision Making (pp. 7–14). Springer.

9. Shafronenko, A. Y., Bodyanskiy, Y. V., & Klymova, I. (2019). Neuro-fuzzy clustering of distorted data using cat swarm optimization. LAP LAMBERT Academic Publishing.

10. Bodyanskiy, Y., & Shafronenko, A. (2021). Adaptive recovery of distorted data based on credibilistic fuzzy clustering approach. In COLINS (pp. 6–15).

11. Bodyanskiy, Y., Shafronenko, A., & Pliss, I. (2021). Adaptive fuzzy probabilistic clustering of incomplete data. International Journal "Information. Models & Analyses", 2(2), 112–117.

12. Bodyanskiy, Y., Shafronenko, A., & Mashtalir, S. (2019). Online robust fuzzy clustering of data with omissions using similarity measure of special type. Springer, 13.

13. Bodyanskiy, Y., Shafronenko, A., & Klymova, I. (2021). Online fuzzy clustering of incomplete data using credibilistic approach and similarity measure of special type. Radio Electronics, Computer Science, Control, 97, 97–104.

14. Mashtalir, S., & Bodyanskiy, Y. V. (2019). Adaptive fuzzy clustering for incomplete data in dynamic systems. System Research and Information Technologies, 11, 25–33.

15. Volkova, V. V., & Bodyanskiy, Y. V. (2011). Neuro-fuzzy clustering of distorted data in incomplete datasets. Inductive Modelling of Complex Systems, 6, 13–21.

16. Shafronenko, A., Bodyanskiy, Y., & Pliss, I. (2019). Wavelet analysis and decomposition into color spaces in researching human fluorescently labeled tissue images. In 2019 IEEE International Conference on Advanced Optoelectronics.

17. Bodyanskiy, Y., Shafronenko, A., & Pliss, I. (2020). Adaptive fuzzy clustering of incomplete data. Science and Education a New Dimension. Natural and Technical Sciences, 193, 112–117.

18. Volkova, V. V., & Bodyanskiy, Y. V. (2011). Neuro-fuzzy clustering of data with missing values. Inductive Modelling of Complex Systems, 7, 23–31.

19. Bodyanskiy, Y. V., & Shafronenko, A. (2019). Adaptive recovery of distorted data using neuro-fuzzy approaches. Artificial Intelligence and Applications, 19, 14–21.

20. Klymova, I., & Shafronenko, A. (2020). Modified fuzzy clustering algorithms for dynamic data analysis. In Proceedings of the International Conference on Advanced Data Analytics (pp. 33–41).

21. Kasatkina, N. V., & Bodyanskiy, Y. V. (2020). Fuzzy clustering in data streams using similarity measures of special types. Radio Electronics, Computer Science, Control, 58, 25–30.

22. Kasatkina, N. V., & Bodyanskiy, Y. V. (2021). Real-time clustering of sensor networks using fuzzy methods. Journal of Advanced Computational Systems, 18, 12–19.

23. Volkova, V. V., & Mashtalir, S. (2018). Neuro-fuzzy networks for adaptive data clustering. Artificial Intelligence Applications, 25, 32–40.

24. Mashtalir, S., & Bodyanskiy, Y. V. (2022). Analysis of large-scale datasets using adaptive clustering techniques. In Lecture Notes in Computational Intelligence (pp. 432–234).

25. Shafronenko, A., Bodyanskiy, Y., & Volkova, V. (2013). Adaptive neuro-fuzzy clustering of data with omissions. In East West Fuzzy Colloquium Proceedings, 20, 214–223.

26. Rudenko, D. A., & Bodyanskiy, Y. V. (2020). Credibilistic approaches for clustering with incomplete data. Journal of Computational Techniques and Systems Analysis, 22, 45–52.

27. Pliss, I. P., & Bodyanskiy, Y. V. (2022). Evolutionary optimization methods for fuzzy clustering. System Research and Information Technologies, 79, 87–95.

28. Bodyanskiy, Y., Shafronenko, A., & Mashtalir, S. (2022). Adaptive fuzzy clustering methods for analyzing large data arrays. In Lecture Notes on Intelligent Systems (pp. 432–439).

29. Shafronenko, A. (2022). Adaptive fuzzy clustering approach for missing data. Journal of Advanced Computational Techniques, 13(3), 21–33.

30. Bodyanskiy, Y., Shafronenko, A., & Klymova, I. (2019). Neuro-fuzzy clustering of distorted data using cat swarm optimization. LAP LAMBERT Academic Publishing.

31. Shafronenko, A., Bodyanskiy, Y., & Pliss, I. (2023). Adaptive clustering of incomplete data based on modified algorithms. Journal of Computational Research, 10(5), 122–128.

32. Shafronenko, A., Bodyanskiy, Y., & Rudenko, D. (2020). Adaptive neuro-fuzzy methods for distorted data clustering. Radio Electronics, Computer Science, Control, 92–99.

33. Bodyanskiy, Y., Shafronenko, A., & Klymova, I. (2020). Adaptive neuro-fuzzy clustering of data with omissions. Artificial Intelligence Applications, 17, 12–21.

34. Bodyanskiy, Y., & Shafronenko, A. (2021). Adaptive clustering algorithms for incomplete data using neuro-fuzzy techniques. Artificial Intelligence Systems, 19(2), 14–23.

35. Volkova, V. V., & Bodyanskiy, Y. V. (2012). Adaptive clustering approaches for missing data. Artificial Intelligence Trends, 23, 56–64.

36. Bodyanskiy, Y. V., & Rudenko, D. A. (2021). Adaptive clustering algorithms for data streams using density-based approaches. Journal of Data Science and Applications, 14, 67–73.

37. Mashtalir, S., & Bodyanskiy, Y. V. (2019). Adaptive clustering for incomplete data in dynamic systems. Artificial Intelligence Journal, 27, 45–55.

38. Кудрявський А. (2024). Особливості застосування методів кластеризації на основі щільності розподілу потоків даних. VII Всеукраїнська студентська наукова конференція, 22 Листопада, 2024, Львів , С. 446-447.